

SOME 3CNF PROPERTIES ARE HARD TO TEST*

ELI BEN-SASSON[†], PRAHLADH HARSHA[‡], AND SOFYA RASKHODNIKOVA[§]

Abstract. For a Boolean formula φ on n variables, the associated property P_φ is the collection of n -bit strings that satisfy φ . We study the query complexity of tests that distinguish (with high probability) between strings in P_φ and strings that are far from P_φ in Hamming distance. We prove that there are 3CNF formulae (with $O(n)$ clauses) such that testing for the associated property requires $\Omega(n)$ queries, even with adaptive tests. This contrasts with 2CNF formulae, whose associated properties are always testable with $O(\sqrt{n})$ queries [E. Fischer et al., *Monotonicity testing over general poset domains*, in Proceedings of the 34th Annual ACM Symposium on Theory of Computing, ACM, New York, 2002, pp. 474–483]. Notice that for every negative instance (i.e., an assignment that does not satisfy φ) there are three bit queries that witness this fact. Nevertheless, finding such a short witness requires reading a constant fraction of the input, even when the input is very far from satisfying the formula that is associated with the property.

A property is *linear* if its elements form a linear space. We provide sufficient conditions for linear properties to be hard to test, and in the course of the proof include the following observations which are of independent interest:

1. In the context of testing for linear properties, adaptive two-sided error tests have no more power than nonadaptive one-sided error tests. Moreover, without loss of generality, any test for a linear property is a linear test. A linear test verifies that a portion of the input satisfies a set of linear constraints, which define the property, and rejects if and only if it finds a falsified constraint. A linear test is by definition nonadaptive and, when applied to linear properties, has a one-sided error.
2. Random low density parity check codes (which are known to have linear distance and constant rate) are not locally testable. In fact, testing such a code of length n requires $\Omega(n)$ queries.

Key words. sublinear algorithms, lower bounds, property testing, CNF formulae, locally testable codes

AMS subject classification. 68Q17

DOI. 10.1137/S0097539704445445

1. Introduction. Property testing deals with a relaxation of decision problems, where one must determine whether an input belongs to a particular set, called a *property*, or is far from it. “Far” usually means that many characters of the input have to be modified to obtain an element in the set. Property testing was first formulated by Rubinfeld and Sudan [RS96] in the context of linear functions and was

*Received by the editors July 30, 2004; accepted for publication (in revised form) March 17, 2005; published electronically September 8, 2005. A preliminary version of this paper appeared in the *Proceedings of the 35th Annual ACM Symposium on Theory of Computing*, 2003 [BHR03].

<http://www.siam.org/journals/sicomp/35-1/44544.html>

[†]Computer Science Department, Technion - Israel Institute of Technology, Haifa, Israel (eli@eecs.harvard.edu; elli@cs.technion.ac.il). This work was done while the author was a postdoctoral researcher at the Department of Engineering and Applied Sciences, Harvard University, Cambridge, MA 02138, and Laboratory for Computer Science, Massachusetts Institute of Technology, Cambridge, MA 02139. This author’s work was supported by NSF grants CCR 0133096, CCR 9877049, CCR 9912342, and CCR 0205390, and by NTT Award MIT 2001-04.

[‡]Microsoft Research, 1065 La Avenida, Mountain View, CA 94043 (pharsha@microsoft.com). This work was done at the Laboratory for Computer Science, Massachusetts Institute of Technology, Cambridge, MA 02139 and was supported in part by NSF Award CCR 9912342 and NTT Award MIT 2001-04.

[§]Weizmann Institute of Science, Rehovot, Israel (sofya@theory.csail.mit.edu). This work was done at the Laboratory for Computer Science, Massachusetts Institute of Technology, Cambridge, MA 02139.

applied to combinatorial objects, especially graphs, by Goldreich, Goldwasser, and Ron [GGR98]. Property testing has recently become quite an active research area; see [Ron01, Fis01] for surveys on the topic.

One of the important problems in property testing is characterizing properties that can be tested with a sublinear¹ number of queries to the input (cf. [GGR98, AKNS01, New02, FN04, AFKS00, Fis01, GT03]; see also section 1.1 for more information). Our paper continues this line of research by trying to relate the testability of properties over the binary alphabet with their formula complexity. We prove a linear lower bound for testing some properties with very small formula complexity, thus showing that the formula complexity of the property does not always help to assess the testability of the property. In section 1.1 several strong lower bounds on the query complexity [GGR98, GT03, BOT02, GR02] are discussed and compared to our lower bound.

Testing k CNFs. A *property* is a collection of strings of a fixed size n . Every property over the binary alphabet can be represented by a conjunctive normal form (CNF) formula, whose set of satisfying assignments equals the set of strings in the property. Testing this property can be viewed as testing whether a given assignment to Boolean variables of the corresponding CNF is close to one that satisfies the formula.² Goldreich, Goldwasser, and Ron [GGR98] prove that there exist properties over the binary alphabet that require testing algorithms to read a linear portion of the input. This implies that testing assignments to general CNF formulae is hard. A natural question is whether restricting CNF formulae to a constant number of variables, k , per clause allows for faster testers. Observe that the standard reduction from satisfiability (SAT) to 3SAT does not apply because it introduces auxiliary variables and thus changes the testing problem.

At first glance, there seems to be hope of obtaining good testers for every fixed k because, for any assignment that does not satisfy the formula, there exists a set of k queries that witnesses this fact. Indeed, Fischer et al. [FLN⁺02] prove that properties expressible as sets of satisfying assignments to 2CNF formulae are testable with $O(\sqrt{n})$ queries, where n is the length of the input. However, we will show that, already for $k = 3$, testing whether an input assignment is close to satisfying a fixed k CNF formula might require a linear number of queries.

Results and techniques. We show the existence of families of 3CNF formulae over n variables (for arbitrarily large n) of size $O(n)$ such that the corresponding properties are not testable with $o(n)$ queries. Thus, we present a gap between 2CNFs and 3CNFs. Our lower bound applies to *adaptive* tests, i.e., tests in which queries might depend on the answers to previous queries. This gives a class of properties which are easy to decide in the standard sense³ but are hard to test.

Each hard 3CNF property we use is a linear⁴ space $V \subseteq \{0, 1\}^n$ that can be expressed as the set of solutions to a set of homogeneous linear constraints of weight 3 (i.e., a 3LIN formula). While proving the lower bound, we show that every adaptive

¹We measure the query complexity as a function of the input length. Thus, linear (sublinear, respectively) query complexity means query complexity that is linear (sublinear) in the input length.

²Our problem should not be confused with the problem of testing whether a CNF formula is “close” to being satisfiable (under a proper definition of closeness). In our problem the CNF formula is fixed and known to the tester. See section 1.1 for a discussion of the seemingly related problem.

³A property is easy to decide in the following standard sense: given the entire assignment, it can be checked in time linear in the number of variables and size of the formula, whether or not the assignment is a satisfying one.

⁴We work over the field with two elements and our linear space is defined over this field.

two-sided error test for checking membership in a vector space can be converted to a nonadaptive one-sided error test with the same query complexity and essentially identical parameters. This allows us to consider only one-sided error nonadaptive tests. In order to prove that a particular linear space V is hard, we need to find, for every such test T , a *bad* vector $b \in \{0, 1\}^n$ (that is, far from V) such that T accepts b with significant probability (i.e., T fails to reject b , as it should). Yao's minimax principle allows us to switch the quantifiers. In other words, in order to prove our lower bound, it suffices to present a distribution \mathcal{B} over bad vectors such that any deterministic test (making few queries) fails to reject a random b (selected according to the distribution \mathcal{B}) with significant probability.

We now give a rough picture of how to get a vector space V that is hard to test and a distribution \mathcal{B} that shows this hardness (per Yao's minimax principle). Fix $0 < \kappa < 1$ and let V be the set of solutions to a system \mathcal{A} of $m = \kappa n$ random linear constraints over n Boolean variables, where each constraint is the sum of a *constant* number of randomly selected variables, and each variable appears in a *constant* number of constraints. Such linear spaces are called random low density parity check (LDPC) codes.⁵ These codes were introduced by Gallager [Gal63], who showed that they have constant rate and (with high probability) large minimal distance. It is possible to show that with high probability the random constraints are linearly independent. Our bad distribution \mathcal{B} is the uniform distribution over vectors that satisfy all but one random constraint of \mathcal{A} . Since the constraints are linearly independent, this distribution is well defined. By definition, each input chosen according to \mathcal{B} is not in the property. It is less obvious, but still true, that each such input is *far* from the property. The tricky part is to show that every deterministic test making $o(n)$ queries will fail to reject a random input chosen according to \mathcal{B} .

A natural way to test if an input belongs to V is to select a few random constraints in \mathcal{A} , query all entries lying in their supports, and accept if and only if all constraints are satisfied. This test always accepts inputs in V . It correctly rejects an input distributed according to \mathcal{B} when the unique random constraint falsified by the input is queried. This method is costly in query complexity because there are $O(n)$ constraints, and only one randomly chosen constraint is not satisfied. A more efficient way to attack the distribution \mathcal{B} would be to use linearity, as follows. If an input satisfies a set of constraints, it must satisfy their sum; if it falsifies exactly one of the constraints in the sum, it must falsify the sum. Thus, one might choose a set of constraints in \mathcal{A} , take their sum, and query the entries in the support of the sum. The summation might cancel out some entries (namely, those that appear in an even number of summands) and reduce the query complexity. This suggests the following general test for testing membership in V : query the input in a small subset of entries and accept if and only if these entries satisfy all possible sums (of constraints in \mathcal{A}), whose support lies entirely within the small subset. In fact, it can be easily observed that any nonadaptive one-sided error test for membership in V is of the above form. Furthermore, we prove that, without loss of generality, the only possible tests (for membership in a linear space) are of the above mentioned form (see Theorem 3.3). The crux of our proof consists of showing that this general method does not significantly reduce the query complexity. Namely, we show that the sum of *any* large subset of the constraints of \mathcal{A} has large support, and thus results in large query complexity. The reason for this phenomenon is the underlying *expansion* of the random set of constraints. Thus, any deterministic

⁵LDPC codes are linear codes defined as solutions to a system of linear constraints with small support.

testing algorithm making $o(n)$ queries essentially checks only $o(m)$ constraints of \mathcal{A} , and thus will reject a random input from \mathcal{B} with subconstant probability.

Connections to coding theory. Our results shed some light on the question of optimal locally testable codes. An infinite family of codes $\{\mathcal{C}\}_n$ is called *locally testable* if the property \mathcal{C}_n is testable with constant query complexity. These codes play a vital role in probabilistically checkable proof (PCP) constructions and are of fundamental importance in theoretical computer science. Recently, Ben-Sasson et al. [BSVW03, BGH⁺04], following the work of Goldreich and Sudan [GS02], proved the existence of such codes, which achieve linear distance and near constant rate, resulting in better PCP constructions.

As mentioned earlier, the vector spaces we use (which are hard to test) are built upon random LDPC codes, which are heavily studied in coding theory (cf. [Gal63] and [Spi95, Chap. 2] and references therein). It follows from an intermediate step in our proof that this important class of codes is not locally testable. Moreover, the property that makes random codes so good in terms of minimal distance, namely expansion, is also behind the poor testability of these codes. In his thesis, Spielman informally discusses why expander codes might not be locally testable and states that a high level of redundancy among the constraints of the code might be required to make it testable ([Spi95, Chap. 5]). In our proof, we make this argument formal and prove that random (c, d) -regular LDPC codes are not locally testable (see Theorem 3.7). This sheds some light on the question of optimal locally testable codes. The existence of such optimal codes that (i) achieve constant rate, (ii) achieve linear distance, and (iii) are locally testable (or even testable with a sublinear number of queries) remains an interesting open problem.

1.1. Connection to previous results.

Classes of testable properties. One of the important problems in property testing is characterizing properties that can be tested with a sublinear number of queries to the input. A series of works identified classes of properties testable with constant query complexity. Goldreich, Goldwasser, and Ron [GGR98] found many such properties. Alon et al. [AKNS01] proved that all regular languages are testable with constant complexity. Newman [New02] extended their result to properties that can be computed by oblivious read-once constant-width branching programs. Fischer and Newman [FN04] demonstrated a property that requires superconstant query complexity and is computable by a read-twice constant-width branching program, thus showing that Newman’s result does not generalize to read-twice branching programs. Several papers [AFKS00, Fis05] worked on the logical characterization of graph properties testable with a constant number of queries. Goldreich and Trevisan [GT03] provide a characterization of properties testable with a constant number of queries and one-sided error in the framework of graph partition properties.

Linear lower bounds. The published linear lower bounds are the aforementioned generic bound due to Goldreich, Goldwasser, and Ron [GGR98], later extended by Goldreich and Trevisan [GT03] to monotone graph properties in NP, and the bound for testing 3-coloring in bounded degree graphs due to Bogdanov, Obata, and Trevisan [BOT02]. In addition, there is a simple and elegant unpublished linear lower bound, observed by Madhu Sudan in a personal communication to the authors. His property consists of polynomials of degree at most $n/2$ over a finite field \mathbb{F}_n of size n , where each polynomial is given by its evaluation on all elements of the field. It is not hard to see that every nonadaptive one-sided error test for this property requires linear query complexity. Since the property of low-degree polynomials is linear, our

reduction from general to nonadaptive one-sided error tests implies a linear lower bound for adaptive two-sided tests for this property. A related property, suggested by Oded Goldreich in a personal communication to the authors, consists of a random linear code. It is not hard to show, using similar reasoning, that with high probability testing this property requires linear query complexity. Observe that both of these properties (low degree polynomials and random codes) are easy to decide once all the input is read, but both cannot be represented by a family of 3CNF formulae.⁶

The aforementioned linear lower bounds of Sudan and Goldreich and of Bogdanov, Obata, and Trevisan capitalize on the existence of inputs that are far from having the property, yet *any* local view of a constant fraction of them can be extended to an element having the property.⁷ But if the property is defined by a k CNF φ , this cannot happen. Clearly, any string that does not have the property must falsify at least one clause of φ . Thus, there is some view of the input of size k that proves that the input does not have the property. Our result shows that, in certain cases, finding such a falsified clause requires reading a constant fraction of the input, even if the assignment is far from any satisfying one. A similar phenomenon is exhibited by Goldreich and Ron for testing bipartiteness in 3-regular, n -vertex graphs [GR02]. They showed a lower bound of $\Omega(\sqrt{n})$ on the query complexity; despite this, short witnesses of nonbipartiteness do exist in the form of odd cycles of length $\text{poly}(\log n)$. Our result strengthens this finding, since in our case the query complexity is linear, whereas the witness size is constant.

Testing an input k CNF. A related problem, but very different from ours, is that of testing whether an *input* k CNF formula is satisfiable. (Recall that in our setting the input is an assignment to a fixed k CNF formula.) The exact version of this problem is a classical NP-complete problem. The property testing version was studied by Alon and Shapira [AS03]. They showed that satisfiability of k CNF formulae is testable with complexity independent of the input size.⁸ In contrast, our problem is very easy in its exact version but hard in its property testing version for $k \geq 3$.

2. Definitions.

Property testing. A *property* is a collection of strings of a fixed size n . A property is *linear* if it forms a vector space (over some underlying field). In this paper, strings are over a binary alphabet unless mentioned otherwise. The distance $\text{dist}(x, \mathcal{P})$ of a string x to a property \mathcal{P} is $\min_{x' \in \mathcal{P}} \text{dist}(x, x')$, where $\text{dist}(x, x')$ denotes the Hamming distance between the two strings x and x' . The *relative distance* of x to \mathcal{P} is its distance to \mathcal{P} divided by n . A string is ε -far from \mathcal{P} if its relative distance to \mathcal{P} is at least ε .

A *test for property \mathcal{P} with distance parameter ε , positive error η_+ , negative error η_- and query complexity q* is a probabilistic algorithm that queries at most q bits of the input, accepts strings in \mathcal{P} with probability at least $1 - \eta_+$, and accepts strings that are ε -far from \mathcal{P} with probability at most η_- , for some $0 \leq \eta_- < 1 - \eta_+ \leq 1$.

⁶In other words, these properties cannot be decided by a family of circuits of depth 2, where the output gate of the circuit is an AND-gate, the next level of gates are all OR-gates of fan-in 3, and the inputs to the OR-gates are either the input bits or their negations.

⁷For example, in Sudan's construction any evaluation of a polynomial on d points can be extended to an evaluation of a polynomial of degree $d' > d$. Thus, $n/2$ values of the polynomial cannot prove or disprove that the polynomial has degree at most $n/2$.

⁸Complexity of a testing problem depends on the definition of distance; in Alon and Shapira's work the distance from the input to a satisfiable formula is defined as the number of clauses that have to be removed to make the input formula satisfiable.

Note that the positive error η_+ is the maximum error made by the test on the YES-instances (i.e., strings in \mathcal{P}), and the negative error η_- is the maximum error made on NO-instances (i.e., strings ε -far from \mathcal{P}). Sometimes we refer to $\eta_+ + \eta_-$ as the sum of errors made by the test T . A test is said to have *error* η if $\eta_+, \eta_- \leq \eta$ (for $\eta < \frac{1}{2}$). If a test T accepts input x , we say $T(x) = 1$. Otherwise, we say $T(x) = 0$. A test with distance parameter ε and error η is referred to as an (ε, η) -test (analogously, $(\varepsilon, \eta_+, \eta_-)$ -test). An ε -test denotes a test with distance parameter ε . A property is (ε, η, q) -testable if it has an (ε, η) -test that makes at most q queries on every input; $(\varepsilon, \eta_+, \eta_-, q)$ -testable is defined analogously.

Two special classes of tests are of interest. An algorithm is *nonadaptive* if it makes all queries in advance before getting the answers. Namely, a query may not depend on the answers to previous queries. An algorithm has a *one-sided error* if it always accepts an input that has the property. In other words, an algorithm has a one-sided error if the positive error η_+ is 0.

CNF and linear formulae. Recall that a Boolean formula is in CNF if it is a conjunction of clauses, where every clause is a disjunction of literals. (A literal is a Boolean variable or a negated Boolean variable.) If all clauses contain at most three literals, the formula is a 3CNF.

A *linear* (LIN) Boolean formula is a conjunction of constraints, where every constraint is satisfied if and only if the variables in the constraint add up to 0 mod 2. If all constraints contain at most d literals, the formula is a d LIN.

Let φ be a formula on n variables. An n -bit string *satisfies* φ if it satisfies all clauses (constraints) of the formula. An n -bit string is ε -far from satisfying φ if at least an ε fraction of the bits needs to be changed to make the string satisfy φ . Each formula φ defines a property $\{x \mid x \text{ satisfies } \varphi\}$. For brevity, we refer to a test for this property as a test for φ .

Random regular bipartite graphs and LDPC codes. Let $G = \langle L, R, E \rangle$ be a bipartite multigraph, with $|L| = n, |R| = m$, and let $d(v)$ be the degree of a vertex v . G is called (c, d) -regular if for all $v \in L$, $d(v) = c$, and if for all $v \in R$, $d(v) = d$. A random (c, d) -regular graph with n left vertices and $m = \frac{c}{d}n$ right vertices⁹ is obtained by selecting a random matching between cn “left” nodes labeled $\{v_1^1, \dots, v_1^c, v_2^1, \dots, v_n^c\}$ and $dm = cn$ “right” nodes labeled $\{u_1^1, \dots, u_m^d\}$, collapsing every c consecutive nodes on the left to obtain n c -regular vertices, and collapsing every d consecutive nodes on the right to obtain m d -regular vertices. Formally, let $L = \{v_1, \dots, v_n\}$, $R = \{u_1, \dots, u_m\}$, and connect vertex v_i to u_j if and only if there exist $\alpha \in [c], \beta \in [d]$ such that (v_i^α, u_j^β) is an edge of the random matching. Notice that the resulting graph may be a multigraph (i.e., have multiple edges between two vertices). The code associated with G , called an LDPC code, was first described and analyzed by Gallager [Gal63].

DEFINITION 2.1 (LDPC Code). *Let $G = \langle L, R, E \rangle$ be a bipartite multigraph with $|L| = n, |R| = m$. Associate a distinct Boolean variable x_i with any $i \in L$. For each $j \in R$, let $N(j) \subseteq L$ be the set of neighbors of j . The j th constraint is $A_j(x_1, \dots, x_n) = \sum_{i \in N(j)} x_i \pmod{2}$. (Notice that a variable may appear several times in a constraint because G is a multigraph.) Let $\mathcal{A}(G)$ be the d LIN formula*

⁹Typically, one fixes c, d to be constants, whereas n, m are unbounded.

$\mathcal{A}(G) = \bigwedge_{j=1}^m (A_j(x) = 0)$. The code defined by G is the property defined by $\mathcal{A}(G)$, namely,

$$\mathcal{C}(G) = \{x \in \{0, 1\}^n \mid \forall j \in [m] \ A_j(x) = 0\}.$$

A random (c, d) -regular LDPC code of length n is obtained by taking $\mathcal{C}(G)$ for a random (c, d) -regular graph G with n left vertices.

3. Main theorem. In this section we state and prove the main theorem and show that some 3CNF properties are hard to test.

THEOREM 3.1 (main). *There exist $0 < \delta^*, \varepsilon^*, \eta^* < 1$ such that, for every sufficiently large n , there is a 3CNF formula φ on $n^* = \Theta(n)$ variables with $\Theta(n)$ clauses such that every adaptive $(\varepsilon^*, \eta_+, \eta_-, q)$ -test for φ with the sum of errors $\eta_+ + \eta_- \leq \eta^*$ makes at least $q = \delta^* n^*$ queries.*

To prove Theorem 3.1, we need to find 3CNF formulae that define hard properties. Our main idea is to work with linear properties (i.e., vector spaces). We prove that, for linear properties, tests of a very simple kind, which we call *linear*, are as powerful as general tests. In particular, linear tests are nonadaptive and have a one-sided error. Working with linear properties allows us to focus on proving a lower bound for this simple kind of tests—bypassing often insurmountable issues of adaptivity and two-sided error.

To explain how we find 3CNF formulae that define hard *linear* properties, we need some linear algebra terminology. Let \mathbb{F} be a field. We say that two vectors $u, v \in \mathbb{F}^n$ are orthogonal to each other (denoted $u \perp v$) if $\sum_{i=1}^n u_i \cdot v_i = 0$. Furthermore, we say that a vector u is orthogonal to a subset $S \subseteq \mathbb{F}^n$ (denoted $u \perp S$) if $u \perp v$ for all vectors $v \in S$. For a linear space $V \subseteq \mathbb{F}^n$ over the field \mathbb{F} , the *dual space* V^\perp is defined as the set of all vectors orthogonal to V (i.e., $V^\perp \triangleq \{u \in \mathbb{F}^n : u \perp V\}$). For $I \subseteq [n]$, let V_I^\perp be the subset of V^\perp composed of all vectors with support in I (i.e., $u \in V_I^\perp$ if and only if $u \in V^\perp$ and the indices of nonzero entries of u lie in I).

DEFINITION 3.2 (linear test). *A test for a linear property $V \subseteq \mathbb{F}^n$ is called a linear test if it is performed by selecting $I = \{i_1, \dots, i_q\} \subseteq [n]$ (according to some distribution), querying w at coordinates I , and accepting if and only if $w \perp V_I^\perp$.*

Linear tests are by definition nonadaptive and have only a one-sided error (members of V are always accepted). Since the inception of property testing, linear properties have been invariably tested by linear tests (starting with [BLR93]). The following theorem shows this is not a coincidence.

THEOREM 3.3 (linear properties have linear tests). *If a linear property $V \subseteq \mathbb{F}^n$ over a finite field \mathbb{F} has a two-sided error adaptive $(\varepsilon, \eta_+, \eta_-, q)$ -test, then it has a linear $(\varepsilon, 0, \eta_+ + \eta_-, q)$ -test.*

The proof of Theorem 3.3 appears in section 5. The reduction to linear tests does not increase the overall error but rather shifts it from the YES-instances to the NO-instances, maintaining the sum of errors $\eta_+ + \eta_-$. Although stated for general finite fields, this theorem is used in our paper only for linear properties over the binary alphabet, namely, with $\mathbb{F} = GF(2)$.

Consider a vector space $V \subseteq GF(2)^n$ and let $\mathcal{A} = (A_1, \dots, A_m)$ be a basis for the dual space V^\perp . Denote the i th coordinate of $x \in GF(2)^n$ by x_i . For two vectors $x, y \in GF(2)^n$, let $\langle x, y \rangle = \sum_{i=1}^n x_i y_i \pmod{2}$. We can view each vector $A_i \in \mathcal{A}$ as a linear constraint on Boolean variables x_1, \dots, x_n of the form $\langle x, A_i \rangle = 0$. This gives us a way to see a vector space as a set of vectors satisfying all constraints in the dual

space, or, equivalently, in the basis of the dual space: $V = \{x \mid \langle x, A_i \rangle = 0 \text{ for all } A_i \in \mathcal{A}\}$. Linear constraints can be thought of as linear formulae.

Let $|x|$ denote the size of the support of vector $x \in GF(2)^n$. Viewing each A_i as a *constraint*, we can represent V as a d LIN formula, where $d = \max_{A_i \in \mathcal{A}} |A_i|$. We work with an arbitrary constant d and later show how to reduce it to 3. Since each 3LIN formula has an equivalent 3CNF, to prove Theorem 3.1 it is enough to find hard 3LINS.

We now present sufficient conditions for a vector space to be hard to test. To understand the conditions, keep in mind that later we employ Yao's minimax principle to show that all vector spaces satisfying these conditions are hard for linear tests. Yao's principle implies that to prove that each low-query *probabilistic* linear test fails on some input, it is enough to give a distribution on the inputs on which each low-query *deterministic* linear test fails. Therefore, we need to exhibit a distribution on vectors that are far from the vector space but are orthogonal with high probability to any fixed set of linear constraints that have support $\leq q$.

DEFINITION 3.4 (hard linear properties). *Let $V \subseteq GF(2)^n$ be a vector space and let \mathcal{A} be a basis for V^\perp . Fix $0 < \varepsilon, \mu < 1$.*

- \mathcal{A} is ε -separating if every $x \in GF(2)^n$ that falsifies exactly one constraint in \mathcal{A} has $|x| \geq \varepsilon n$.
- \mathcal{A} is (q, μ) -local if every $\alpha \in GF(2)^n$ that is a sum of at least μm vectors in \mathcal{A} has $|\alpha| \geq q$.

For the proof that every vector space satisfying the above conditions is hard to test, our bad distribution that foils low-query tests is over strings that falsify exactly one constraint. The falsified constraint is chosen uniformly at random. The first condition ensures that the distribution is over vectors which are ε -far from the vector space. (To see this, notice that if the distance of x from $y \in V$ is less than εn , then $|x + y| < \varepsilon n$ and $x + y$ falsifies exactly one constraint, contradicting the first condition.)

The second condition ensures that the distribution is hard to test. Assume that each deterministic linear test corresponds to some vector $u \in V^\perp, |u| \leq q$. (This is oversimplified because a deterministic linear test may read several dual vectors, whose combined support size is at most q . However, this simple case clarifies our approach and is not far from the formal proof given in section 4.) Vector u can be expressed as a linear combination of vectors in the basis: $u = \sum_{j \in J} A_j$ for some $J \subset [m]$. Let A_k be the (random) constraint falsified by a vector w in our hard distribution. Clearly, u will reject w if and only if $k \in J$. The second condition implies that this will occur with probability at most μ . This intuitive discussion is formalized by the following theorem, proved in section 4.

THEOREM 3.5. *Let $V \subseteq GF(2)^n$ be a linear space. If V^\perp has an ε -separating (q, μ) -local basis $\mathcal{A} = (A_1, \dots, A_m)$ and $0 < \mu < 1/2$, then every linear ε -test for it with error $\leq 1 - 2\mu$ requires q queries.*

We now turn to constructing linear spaces that are hard to test. In particular, we show that for sufficiently large constants c, d , with high probability a random (c, d) -regular LDPC code (per Definition 2.1) is hard according to Definition 3.4. The proof of this lemma, which uses the probabilistic method, appears in section 6. (We do not attempt to optimize constants.)

LEMMA 3.6 (hard linear properties exist). *Fix odd integer $c \geq 7$ and constants*

$\mu, \varepsilon, \delta, d > 0$ satisfying

$$\mu \leq \frac{1}{100} \cdot c^{-2}; \quad \delta < \mu^c; \quad d > \frac{2\mu c^2}{(\mu^c - \delta)^2}; \quad \varepsilon \leq \frac{1}{100} \cdot d^{-2}.$$

Then, for all sufficiently large n , with high probability for a random (c, d) -regular graph G with n left vertices, the dLIN formula $\mathcal{A}(G)$ (as in Definition 2.1) is linearly independent, ε -separating, and $(\delta n, \mu)$ -local.

We now have an abundance of dLIN formulae that are hard to test for sufficiently large d . As an immediate corollary, we conclude that random LDPC codes are hard to test.

THEOREM 3.7 (random (c, d) -regular LDPC codes are hard to test). *Let $c, d, \mu, \varepsilon, \delta$ satisfy the conditions of Lemma 3.6. For sufficiently large n , with high probability a random (c, d) -regular LDPC code $\mathcal{C}(G)$ of length n satisfies the following: Every adaptive $(\varepsilon, \eta_+, \eta_-, q)$ -test for $\mathcal{C}(G)$ with the sum of errors $\eta_+ + \eta_- \leq 1 - 2\mu$ makes at least $q = \delta n$ queries.*

Proof. The proof follows directly from Lemma 3.6 and Theorems 3.3 and 3.5. \square

The following reduction brings d down to 3 while preserving the conditions of Definition 3.4 (with smaller constants).

LEMMA 3.8 (reduction to 3CNFs). *Suppose $\mathcal{A} \subseteq \{0, 1\}^n$ is a set of $m = \frac{c}{d}n$ vectors, each vector of weight at most d . Suppose furthermore \mathcal{A} is (i) linearly independent, (ii) ε -separating, and (iii) $(\delta n, \mu)$ -local. Then there exists a set $\mathcal{A}^* \subset \{0, 1\}^{n^*}$ of m^* vectors, each vector of weight at most 3, such that \mathcal{A}^* is (i) linearly independent, (ii) ε^* -separating, and (iii) $(\delta^* n^*, \mu^*)$ -local, for*

$$\begin{aligned} m^* &\leq 2dm; & n &\leq n^* \leq (2c+1) \cdot n; & \varepsilon &\geq \varepsilon^* \geq \frac{\varepsilon}{(2c+1)}; \\ \delta &\geq \delta^* \geq \frac{\delta}{d^{\log d+1} \cdot (2c+1)}; & \mu^* &\leq \mu + \frac{\delta(\log d+1)}{c}. \end{aligned}$$

Lemma 3.8 is proved in section 7. We now complete the proof of our main theorem.

Proof of Theorem 3.1 (main). We start by fixing the following parameters:

$$c = 7; \quad \mu = \frac{1}{100} \cdot c^{-2}; \quad \delta = \frac{\mu^c}{2}; \quad d = \left\lceil \frac{4\mu c^2}{(\mu^c - \delta)^2} \right\rceil = \lceil 16c^2 \mu^{1-2c} \rceil; \quad \varepsilon = \frac{1}{100} \cdot d^{-2}.$$

We pick $\mathcal{A}_n \subset \{0, 1\}^n$ to be a linearly independent, $(\delta n, \mu)$ -local, ε -separating collection of vectors, of weight $\leq d$. By Lemma 3.6, such a set \mathcal{A}_n exists for our setting of $\mu, \varepsilon, \delta, d$ and sufficiently large n .

Next, let $\mathcal{A}_{n^*}^* \subset \{0, 1\}^{n^*}$ be a linearly independent, $(\delta^* n^*, \mu^*)$ -local, ε^* -separating set of vectors of weight at most 3, ensured by Lemma 3.8 (where $\delta^*, \mu^*, \varepsilon^*, n^*$ are as stated in this lemma). Recall that for every 3LIN formula there is an equivalent 3CNF and let φ be the 3CNF formula equivalent to $\mathcal{A}_{n^*}^*$. Moreover, because $m^*, n^* = \Theta(n)$ and each 3LIN constraint translates into a constant number of 3CNF constraints, we conclude that the number of clauses in φ is linear in n^* .

Notice $\delta^*, \varepsilon^*, \mu^* > 0$ because $\delta, \varepsilon, \mu, d > 0$, and $\delta^*, \varepsilon^* < 1$ because $\delta, \varepsilon < 1$. Furthermore, for our setting of constants,

$$\mu^* \leq \mu + \frac{\delta(\log d+1)}{c} = \mu + \frac{\mu^c(\log(16c^2 \mu^{-(2c-1)}) + 1)}{2c} < \frac{1}{2}.$$

Therefore, $0 < 1 - 2\mu^* < 1$. Set $\eta^* = 1 - 2\mu^*$. By Theorem 3.5, every linear ε^* -test for $\mathcal{A}_{n^*}^*$ with error $\leq \eta^*$ requires $\delta^* n^*$ queries. Theorem 3.3 implies that every

adaptive $(\varepsilon^*, \eta_+, \eta_-, q)$ -test for $\mathcal{A}_{n^*}^*$ with $\eta_+ + \eta_- \leq \eta^*$ makes at least $\delta^* n^*$ queries. This completes the proof of our main theorem. \square

4. Lower bounds for linear tests: Proof of Theorem 3.5. We employ Yao's minimax principle. It states that to prove that every q -query randomized linear test fails with probability more than η , it is enough to exhibit a distribution \mathcal{B} on the inputs for which every q -query deterministic linear test fails with probability more than η . For $i = 1, \dots, m$ let \mathcal{B}_i be the uniform distribution over n -bit strings that falsify constraint A_i and satisfy the rest. The distribution \mathcal{B} is the uniform distribution over the \mathcal{B}_i 's. The comment after Definition 3.4 shows that the distribution \mathcal{B} is over strings which are ε -far from V .

A deterministic linear test T is identified by a subset $I \subseteq [n]$, $|I| = q$ and rejects the input w only if w is not orthogonal to V_I^\perp (see Definition 3.2). Write each vector $u \in V^\perp$ in the basis \mathcal{A} as $u = \mathcal{A} \cdot b_u$, where \mathcal{A} is interpreted as the $m \times n$ matrix whose i th row is A_i and $b_u \in GF(2)^m$. Let $J_T = \cup_{V_I^\perp} \text{supp}(b_u)$. We claim T rejects w distributed according to \mathcal{B} if and only if the index of the unique constraint falsified by w belongs to J_T . This is because w is orthogonal to all but one $A_i \in \mathcal{A}$, so the subspace of V^\perp that is orthogonal to w is precisely the span of $\mathcal{A} \setminus \{A_i\}$. Summing up, the probability that T rejects w is precisely $|J_T|/m$. We now give an upper bound on $|J_T|$. Notice that V_I^\perp is a vector space, so the set $V' \triangleq \{b_u : u \in V_I^\perp\}$ is also a vector space (it is the image of the vector space V_I^\perp under the linear map \mathcal{A}^{-1}). Since \mathcal{A} is (q, μ) -local, we know $|b_u| \leq \mu m$ for every $u \in V_I^\perp$. Thus, V' is a vector space over $GF(2)$ in which each element has support size $\leq \mu m$. We claim that $|J_T| = |\cup_{v' \in V'} \text{supp}(v')| \leq 2\mu m$. To see this, pick a uniformly random vector of $v \in V'$. We claim that

$$\mathbb{E}_{v' \in V'} |v| = |\cup_{v'' \in V'} \text{supp}(v'')|/2.$$

This follows from the linearity of expectation and the fact that the projection of V' onto any $j \in \cup_{V'} \text{supp}(v')$ is a linear function, so the expected value of v'_j is $1/2$. Since \mathcal{A} is (q, μ) -local, we know $\mathbb{E}_{v' \in V'} |v'| \leq \mu m$, which means that $|J_T| \leq 2\mu m$. This implies that our deterministic test (reading q entries of w) will detect a violation with probability at most $|J_T|/m \leq 2\mu$. The proof of Theorem 3.5 is complete. \square

5. Reducing general tests to linear ones. In this section we prove Theorem 3.3 by presenting a generic reduction that converts any adaptive two-sided error test for a linear property to a linear test, as in Definition 3.2. We perform this reduction in two stages: we first reduce an adaptive test with two-sided error to an adaptive linear test (Theorem 5.3), maintaining the sum of the positive and negative errors $(\eta_+ + \eta_-)$, and then remove the adaptivity and maintain all other parameters (Theorem 5.6). The second reduction was suggested by Madhu Sudan. We state and prove these reductions for the general case when the linear spaces V considered are over any finite field \mathbb{F} , though we require them only for the case $\mathbb{F} = GF(2)$.

Preliminaries. Any probabilistic test can be viewed as a distribution over deterministic tests, and each deterministic test can be represented by a decision tree. Thus, any test T can be represented by an ordered pair $(\Upsilon_T, \mathcal{D}_T)$, where $\Upsilon_T = \{\Gamma_1, \Gamma_2, \dots\}$ is a set of decision trees and \mathcal{D}_T is a distribution on this set such that on input x , T chooses a decision tree Γ with probability $\mathcal{D}_T(\Gamma)$ and then answers according to $\Gamma(x)$.

We say that a test *detects a violation* if no string in V is consistent with the answers to the queries. By linearity, it is equivalent to having a constraint α in V^\perp

such that $\langle x, \alpha \rangle \neq 0$ for all $x \in \mathbb{F}^n$, which are consistent with the answers to the queries.

Let V be a vector space. For any leaf l of decision tree Γ , let V_l be the set of all vectors in V that are consistent with the answers along the path leading to l . Similarly, for any string $x \in \mathbb{F}^n$, let V_l^x be the set of all vectors in $x + V$ that are consistent with the answers along the path leading to l .

CLAIM 5.1. *Let \mathbb{F} be a finite field and $V \subseteq \mathbb{F}^n$ be a vector space. Let $x \in \mathbb{F}^n$. For any decision tree Γ and a leaf l in Γ , if both V_l and V_l^x are nonempty, then $|V_l| = |V_l^x|$.*

Proof. Let U be the set of all strings in V which have the element 0 in all the positions queried along the path leading to l . Since $0^n \in U$, we have that U is nonempty. Observe that if $u \in U$ and $v \in V_l$, then $u + v \in V_l$. In fact, if $V_l \neq \emptyset$, $V_l = v + U$ for any $v \in V_l$. Hence, $|V_l| = |U|$. Similarly, if $V_l^x \neq \emptyset$, we have that $V_l^x = y + U$ for any $y \in V_l^x$. Hence, $|V_l^x| = |U|$ and the lemma follows. \square

5.1. Reduction from adaptive two-sided to adaptive linear.

DEFINITION 5.2 (adaptive linear test). *A test for a linear property $V \subseteq \mathbb{F}^n$ is called adaptive linear if it is performed by making adaptive queries $I = \{i_1, \dots, i_q\}$ (according to some distribution) to w and accepting if and only if $w \perp V_I^\perp$.*

Notice that adaptive linear tests have a one-sided error: every $w \in V$ is always accepted.

THEOREM 5.3. *Let \mathbb{F} be a finite field and $V \subseteq \mathbb{F}^n$ a vector space. If V has an adaptive $(\varepsilon, \eta_+, \eta_-, q)$ -test T , then it has a (one-sided error) adaptive linear $(\varepsilon, 0, \eta_+ + \eta_-, q)$ -test T' .*

Proof. Let $T = (\Upsilon_T, \mathcal{D}_T)$ be a two-sided error (adaptive) (ε, η, q) -test for V . To convert T to an adaptive linear one, we modify the test so that it rejects if and only if it observes that a constraint in V^\perp has been violated. We say that a leaf l is labeled *optimally* if its label is 0 when the query answers on the path to l falsify some constraint in V^\perp , and its label is 1 otherwise. We relabel the leaves of each tree Γ in Υ_T *optimally* to obtain the tree Γ_{opt} .

Relabeling produces a one-sided error test with unchanged query complexity. However, the new test performs well only on “average.” To get good performance on every string, we randomize the input x by adding a random vector v from V to it and perform the test on $x + v$ instead of x . Now we formally define T' .

DEFINITION 5.4. *Given a two-sided error (adaptive) test T for V , define the test T' as follows: On input x , choose a decision tree Γ according to the distribution \mathcal{D}_T as T does, choose a random $v \in V$, and answer according to $\Gamma_{\text{opt}}(x + v)$.*

Clearly, T' is adaptive linear and has the same query complexity as T . It remains to check that T' has error $\eta_+ + \eta_-$ on negative instances.

For any $x \in \mathbb{F}^n$ and any test T , let ρ_x^T be the average acceptance probability of test T over all strings in $x + V$, i.e., $\rho_x^T = \text{average}_{y \in x+V} (\Pr[T(y) = 1])$. For notational brevity, we denote $\rho_{0^n}^T$, the average acceptance probability of strings in V , by ρ^T . Observe that, for the new test T' , for each input x , $\Pr[T'(x) = 1] = \rho_x^{T'}$.

Claim 5.5 below shows that the transformation to a one-sided error test given by Definition 5.4 increases the acceptance probability of any string not in V by at most $\rho^{T'} - \rho^T$. Notice that all vectors in $x + V$ have the same distance to V . Therefore if x is ε -far from V , then $\rho_x^T \leq \eta_-$. Together with Claim 5.5, it implies that for all vectors x that are ε -far from V , the error is low:

$$\Pr[T'(x) = 1] = \rho_x^{T'} \leq \rho^{T'} - \rho^T + \rho_x^T \leq 1 - (1 - \eta_+) + \eta_- = \eta_+ + \eta_-.$$

This completes the proof of Theorem 5.3 \square

CLAIM 5.5. $\rho^T - \rho_x^T \leq \rho^{T'} - \rho_x^{T'}$ for any vector $x \in \mathbb{F}^n$.

Proof. Let $x \in \mathbb{F}^n$. It is enough to prove that relabeling one leaf l of a decision tree Γ in Υ_T optimally does not decrease $\rho^T - \rho_x^T$. Then we obtain the claim by relabeling one leaf at a time to get T' from T . There are two cases to consider.

Case (i) The path to l falsifies some constraint in V^\perp . Then l is relabeled from 1 to 0. This change preserves ρ^T because it only affects strings that falsify some constraint. Moreover, it can only decrease the acceptance probability for such strings. Therefore, ρ_x^T does not increase. Hence, $\rho^T - \rho_x^T$ does not decrease.

Case (ii) The path to l does not falsify any constraint in V^\perp . Then l is relabeled from 0 to 1. Let V_l and V_l^x , respectively, be the set of vectors in V and $x+V$ that are consistent with the answers observed along the path to l . Thus, every string in $V_l \cup V_l^x$ was rejected before relabeling but is accepted now. The behavior of the algorithm on the remaining strings in V and $x+V$ is unaltered. Hence, the probability ρ^T increases by the quantity $\mathcal{D}_T(\Gamma) \cdot \frac{|V_l|}{|V|}$. Similarly, ρ_x^T increases by $\mathcal{D}_T(\Gamma) \cdot \frac{|V_l^x|}{|V|}$.

It suffices to show that $|V_l| \geq |V_l^x|$. Since the path leading to l does not falsify any constraint, V_l is nonempty. If V_l^x is empty, we are done. Otherwise, both V_l and V_l^x are nonempty, and by Claim 5.1, $|V_l| = |V_l^x|$. \square

5.2. Reduction to linear tests. In this section, we remove the adaptivity from the linear tests. The intuition behind this is as follows: To check if a linear constraint is satisfied, a test needs to query all the variables that participate in that constraint. Based on any partial view involving some of the variables, the test cannot guess if the constraint is going to be satisfied or not until it reads the final variable. Hence, any adaptive decision based on such a partial view does not help.

THEOREM 5.6. *If $V \subseteq \mathbb{F}^n$ is a vector space over a finite field \mathbb{F} that has an adaptive linear $(\varepsilon, 0, \eta, q)$ -test, then it has a (nonadaptive) linear $(\varepsilon, 0, \eta, q)$ -test.*

Proof. Let T be an adaptive linear $(\varepsilon, 0, \eta, q)$ -test for V . Let Υ_T and \mathcal{D}_T be the associated set of decision trees and the corresponding distribution, respectively.

DEFINITION 5.7. *Given an adaptive linear test T for V , define the test T' as follows: On input x , choose a random $v \in V$, query x on all variables that T queries on input v , and reject if a violation is detected; otherwise accept.*

T' makes the same number of queries as T . Moreover, the queries depend only on the random $v \in V$ and not on the input x . Hence, the test T' is nonadaptive. The following claim relates the acceptance probability of T' to the average acceptance probability of T .

CLAIM 5.8. *Let T be an adaptive linear test and T' the nonadaptive version of T (as in Definition 5.7). Then, for any string $x \in \mathbb{F}^n$,*

$$\Pr[T'(x) = 1] = \text{average}_{v \in V} (\Pr[T(x+v) = 1]).$$

Proof. For any decision tree Γ , let $l_1(\Gamma)$ denote the set of leaves in Γ that are labeled 1. For any leaf l in a decision tree Γ , let $\text{var}(l)$ denote the set of variables queried along the path leading to l in the tree Γ . Following the notation of Claim 5.1, let V_l and V_l^x be the set of all vectors in V and $x+V$, respectively, that are consistent with the answers along the path leading to l . Also let I_l^x be a binary variable which is set to 1 if and only if x does not violate any constraint in V^\perp involving only the variables $\text{var}(l)$. Observe that if test T' chooses the decision tree $\Gamma \in \Upsilon_T$ and the vector $v \in V$ such that $v \in V_l$ for some leaf l labeled 1 in the tree Γ , then $I_l^x = 1$ if and only if $T'(x) = 1$.

The quantity “average $_{v \in V} (\Pr[T(x+v) = 1])$ ” can be obtained as follows: First,

choose a decision tree $\Gamma \in \Upsilon_T$ according to the distribution \mathcal{D}_T . Then for each leaf l labeled 1 in Γ , find the fraction of vectors in $x + V$ that follow the path leading to l . The weighted sum of these fractions is $\text{average}_{v \in V} (\Pr[T(x + v) = 1])$. Thus,

$$(5.1) \quad \text{average}_{v \in V} (\Pr[T(x + v) = 1]) = \sum_{\Gamma \in \Upsilon_T} \mathcal{D}_T(\Gamma) \left(\sum_{l \in l_1(\Gamma)} \frac{|V_l^x|}{|V|} \right).$$

Now consider the quantity $\Pr[T'(x) = 1]$. Test T' can be viewed in the following fashion: On input x , T' chooses a random decision tree $\Gamma \in \Upsilon_T$ according to the distribution \mathcal{D}_T . It then chooses a leaf l labeled 1 in Γ with probability proportional to the fraction of vectors $v \in V$ that are accepted along the path leading to l (i.e., $|V_l|/|V|$), queries x on all variables in $\text{var}(l)$, accepts if $I_l^x = 1$, and rejects otherwise. This gives us the following expression for $\Pr[T'(x) = 1]$:

$$(5.2) \quad \Pr[T'(x) = 1] = \sum_{\Gamma \in \Upsilon_T} \mathcal{D}_T(\Gamma) \left(\sum_{l \in l_1(\Gamma)} \frac{|V_l|}{|V|} \cdot I_l^x \right).$$

From (5.1) and (5.2), it suffices to prove that $|V_l^x| = I_l^x \cdot |V_l|$ for all leaves l labeled 1 in order to prove the claim.

Observe that $|V_l|$ is nonempty since l is labeled 1. Hence, by Claim 5.1, $|V_l| = |V_l^x|$ if V_l^x is also nonempty. It now suffices to show that V_l^x is nonempty if and only if $I_l^x = 1$.

Suppose V_l^x is nonempty. Then there exists $y \in x + V$ that does not violate any constraint involving only the variables $\text{var}(l)$. But y and x satisfy the same set of constraints. Hence, x also does not violate any constraint involving only the variables $\text{var}(l)$. Thus, $I_l^x = 1$.

Now, for the other direction, suppose $I_l^x = 1$. Then the values of the variables $\text{var}(l)$ of x do not violate any constraint in V^\perp . Hence, there exists $u \in V$ that has the same values as x for the variables $\text{var}(l)$. Let $v \in V_l$. Then, the vector $x - u + v \in x + V$ has the same values for the variables $\text{var}(l)$ as v . Hence, V_l^x is nonempty. This concludes the proof of the claim. \square

The above claim proves that T' inherits its acceptance probability from T . As mentioned earlier, T' inherits its query complexity from T . Hence T' is a linear $(\varepsilon, 0, \eta, q)$ -test for V . \square

6. Random codes require a linear number of queries. In this section we prove Lemma 3.6. We start by analyzing the expansion properties of random regular graphs.

6.1. Some expansion properties of random regular graphs. To prove that a random $\mathcal{C}(G)$ obeys Definition 3.4 with high probability, we use standard arguments about expansion of the random graph G . We reduce each requirement on $\mathcal{A}(G)$ to a requirement on G and then show that the expansion of a random graph G implies that it satisfies the requirements. We need the following notions of neighborhood and expansion.

DEFINITION 6.1 (neighborhood). *Let $G = \langle V, E \rangle$ be a graph. For $S \subset V$, let*

- (i) $N(S)$ be the set of neighbors of S .
- (ii) $N^1(S)$ be the set of unique neighbors of S , i.e., vertices with exactly one neighbor in S .
- (iii) $N^{\text{odd}}(S)$ be the set of neighbors of S with an odd number of neighbors in S .

Notice that $N^1(S) \subseteq N^{odd}(S)$.

DEFINITION 6.2 (expansion). *Let $G = \langle L, R, E \rangle$ be a bipartite graph with $|L| = n$, $|R| = m$.*

(i) *G is called a (λ, γ) -right expander if*

$$\forall S \subset R, |S| \leq \gamma n, |N(S)| > \lambda \cdot |S|.$$

(ii) *G is called a (λ, γ) -right unique neighbor expander if*

$$\forall S \subset R, |S| \leq \gamma n, |N^1(S)| > \lambda \cdot |S|.$$

(iii) *G is called a (λ, γ) -right odd expander if*

$$\forall S \subset R, |S| \geq \gamma n, |N^{odd}(S)| > \lambda \cdot |S|.$$

Notice that the definitions of an expander and a unique neighbor expander deal with subsets of size *at most* γn , whereas the definition of an odd expander deals with subsets of size *at least* γn . Left expanders (all three of them) are defined analogously by taking $S \subset L$ in Definition 6.2.

Lemmas 6.3 and 6.6 are proved using standard techniques for analysis of expansion of random graphs, such as those appearing in, for example, [CS88, Spi95].

LEMMA 6.3. *For any integers $c \geq 7, d \geq 2$ and sufficiently large n , a random (c, d) -regular graph with n left vertices is with high probability a $(1, \frac{1}{100} \cdot d^{-2})$ -left unique neighbor expander.*

Proof. We need the following claims, the proofs of which will follow.

CLAIM 6.4. *For any integers $c \geq 2, d$, any constant $\alpha < c - 1$, and sufficiently large n , a random (c, d) -regular bipartite graph with n left vertices is with high probability a (α, ε) -left expander for any ε satisfying*

$$(6.1) \quad \varepsilon \leq \left(2e^{(1+\alpha)} \cdot \left(\frac{\alpha d}{c} \right)^{(c-\alpha)} \right)^{-\frac{1}{c-\alpha-1}}.$$

CLAIM 6.5. *Let G be a (c, d) -regular bipartite graph with n left vertices. If G is a (α, ε) -left expander, then G is a $(2\alpha - c, \varepsilon)$ -left unique neighbor expander.*

Set $\alpha = \frac{c+1}{2}$. Then $\frac{c}{2} < \alpha < c - 1$ for $c \geq 7$. Let G be a random (c, d) -regular bipartite graph. By Claim 6.4, with high probability G is an (α, ε) -right expander for any ε satisfying (6.1).

The following inequalities hold for our selection of α and any $c \geq 7$:

$$\frac{(1 + \alpha)}{(c - \alpha - 1)} \leq 3,$$

$$\frac{\alpha}{c} > \frac{1}{2},$$

$$\frac{(c - \alpha)}{(c - \alpha - 1)} \leq 2.$$

Hence, $\varepsilon = \frac{1}{100} \cdot d^{-2}$ satisfies (6.1). Claim 6.5 completes the proof of Lemma 6.3. \square

Proof of Claim 6.4. Let BAD be the event in which the random graph is *not* an expander. This means there is some $S \subset L, |S| \leq \varepsilon n$ such that $|N(S)| \leq \alpha \cdot |S|$.

Fix sets $S \subset L, T \subset R$, $|S| = s \leq \varepsilon n$, $|T| = \alpha s$, and let B_s be the event in which all edges leaving S land inside T . We upper-bound the probability of this bad event:

$$\Pr[B_s] = \prod_{i=0}^{c \cdot s - 1} \frac{\alpha ds - i}{cn - i} \leq \left(\frac{\alpha ds}{cn} \right)^{cs}.$$

The inequality above holds as long as $\alpha ds < cn$. In the following, we now use a union bound over all sets $S \subset L$, $|S| = s \leq \varepsilon n$ and all sets $T \subset R$, $|T| = \alpha s$. Let κ be the constant $\kappa = e^{1+\alpha} \cdot \left(\frac{\alpha d}{c} \right)^{c-\alpha}$.

$$\begin{aligned} \Pr[BAD] &\leq \sum_{s=1}^{\varepsilon n} \binom{n}{s} \cdot \binom{m}{\alpha s} \cdot \Pr[B_s] \\ &\leq \sum_{s=1}^{\varepsilon n} \left(\frac{en}{s} \right)^s \cdot \left(\frac{em}{\alpha s} \right)^{\alpha s} \cdot \left(\frac{\alpha ds}{cn} \right)^{cs} \\ &= \sum_{s=1}^{\varepsilon n} \left[e^{1+\alpha} \cdot \left(\frac{\alpha d}{c} \right)^{c-\alpha} \cdot \left(\frac{s}{n} \right)^{c-\alpha-1} \right]^s \\ (6.2) \quad &= \sum_{s=1}^{\varepsilon n} \left[\kappa \cdot \left(\frac{s}{n} \right)^{c-\alpha-1} \right]^s. \end{aligned}$$

By definition of α , $c - \alpha - 1 > 0$. Hence $\left(\frac{s}{n} \right)^{c-\alpha-1} \leq 1$. Set

$$(6.3) \quad \varepsilon \leq (2\kappa)^{\frac{-1}{c-\alpha-1}} = \left(2e^{(1+\alpha)} \cdot \left(\frac{\alpha d}{c} \right)^{(c-\alpha)} \right)^{-\frac{1}{c-\alpha-1}}.$$

For this value of ε , each term of the sum (6.2) is at most $\frac{1}{2}$. Set $\lambda = \min\{\frac{1}{3}, \frac{c-\alpha-1}{2}\}$ and split the sum (6.2) into two subsums:

$$\begin{aligned} \Pr[BAD] &\leq \sum_{s=1}^{\varepsilon n} \left[\kappa \cdot \left(\frac{s}{n} \right)^{c-\alpha-1} \right]^s \\ &\leq \sum_{s=1}^{n^\lambda} \left[\kappa \cdot \left(\frac{s}{n} \right)^{c-\alpha-1} \right]^s + \sum_{s=n^\lambda}^{\varepsilon n} \left[\kappa \cdot \left(\frac{s}{n} \right)^{c-\alpha-1} \right]^s \\ &\leq n^\lambda \cdot \kappa \cdot n^{(\lambda-1)2\lambda} + n \cdot 2^{-n^\lambda} \\ &= \kappa \cdot n^{-\lambda+2\lambda^2} + n \cdot 2^{-n^\lambda} \\ &\leq \kappa \cdot n^{-\frac{1}{9}} + n \cdot 2^{-n^\lambda} = o(1). \end{aligned}$$

We conclude that, with high probability, G is an (α, ε) -left expander. \square

Proof of Claim 6.5. Let $S \subset L, |S| \leq \varepsilon|L|$. Then by expansion,

$$\alpha \cdot |S| < |N(S)|.$$

Any neighbor of S that is not a unique neighbor must be touched by at least two edges leaving S . Since the left degree of G is c ,

$$|N(S)| \leq |N^1(S)| + \frac{c \cdot |S| - |N^1(S)|}{2} = \frac{c \cdot |S| + |N^1(S)|}{2}.$$

Combining the two equations, we get our claim. \square

LEMMA 6.6. *For any odd integer c , any constants $\mu > 0, \delta < \mu^c$, and any integer $d > \frac{2\mu c^2}{(\mu^c - \delta)^2}$, a random (c, d) -regular graph is with high probability a (δ, μ) -right odd expander.*

Proof. In the proof, we make use of the following theorem (see [MR95]).

THEOREM 6.7 (Azuma's inequality). *If X_0, \dots, X_t is a martingale sequence such that $|X_i - X_{i+1}| \leq 1$ for all i , then*

$$\Pr[|X_t - X_0| \geq \lambda\sqrt{t}] \leq 2e^{-\lambda^2/2}.$$

Fix $T \subseteq R$ $|T| = t \geq \mu m$. Let $X = |N^{\text{odd}}(T)|$. We start by computing $E[X]$. For $i = 1, \dots, n$, let X_i be the random variable indicating whether vertex $i \in L$ is in $N^{\text{odd}}(T)$. Clearly, $X = \sum_{i=1}^n X_i$, so by the linearity of expectation, we need only compute $E[X_i]$. Recall that $cn = dm$. Let $\text{odd}(c) = \{1, 3, 5, \dots, c\}$ be the set of positive odd integers $\leq c$, and notice that $c \in \text{odd}(c)$ because c is odd:

$$\begin{aligned} E[X_i] &= \frac{\sum_{i \in \text{odd}(c)} \binom{\mu dm}{i} \cdot \binom{(1-\mu)dm}{c-i}}{\binom{cn}{c}} \\ &\geq \frac{\binom{\mu cn}{c}}{\binom{cn}{c}} = \mu^c - O\left(\frac{1}{n}\right). \end{aligned}$$

We conclude by linearity of expectation:

$$E[X] \geq \mu^c \cdot n - O(1).$$

We now use the following edge-exposure martingale to show concentration of X around its expectation. Fix an ordering on the μdm edges leaving T and define a sequence of random variables $Y_0, \dots, Y_{\mu dm}$ as follows: Y_i is the random variable that is equal to the expected size of $N^{\text{odd}}(T)$ after the first i edges leaving T have been revealed. By definition, $Y_{\mu dm} = X$, $Y_0 = E[X]$, and the sequence is a martingale, where $|Y_i - Y_{i+1}| \leq 1$ for all $i \leq \mu dm$. Since $d > \frac{2\mu c^2}{(\mu^c - \delta)^2}$, Azuma's inequality (Theorem 6.7) gives us

$$\begin{aligned} \Pr[X \leq \delta n] &\leq \Pr[|Y_{\mu dm} - Y_0| \geq (\mu^c - \delta)n] \\ &= \Pr\left[|Y_{\mu dm} - Y_0| \geq (\mu^c - \delta)\frac{d}{c}m\right] \\ &\leq 2e^{-\frac{d(\mu^c - \delta)^2}{2\mu c^2} \cdot m} \leq 2e^{-(1+\varepsilon)m}, \end{aligned}$$

where $\varepsilon = \frac{d(\mu^c - \delta)^2}{2\mu c^2} - 1 > 0$. Since there are at most 2^m possible sets $T \subseteq R$, by the union bound,

$$\begin{aligned} \Pr\left[\exists T \subset R \ |T| \geq \mu m, \left|\sum_{j \in T} A_j\right| \leq \delta n\right] &\leq 2^m \cdot 2e^{-(1+\varepsilon)m} \\ &= o(1). \end{aligned}$$

We conclude that with high probability $\mathcal{A}(G)$ is a (δ, μ) -right odd expander. \square

6.2. Proof of Lemma 3.6. Let G be a random (c, d) -regular graph G with n left vertices. We prove that $\mathcal{A}(G)$ is with high probability (i) linearly independent, (ii) $(\delta n, \mu)$ -local, and (iii) ε -separating.

- (i) We need to show that adding up any subset of $\mathcal{A}(G)$ cannot yield $\vec{0}$. Since we are working modulo 2, this is equivalent to proving

$$\forall T \subseteq R, \quad N^{\text{odd}}(T) \neq \emptyset.$$

For small T we use unique neighbor expansion, and for large T we use odd neighbor expansion.

Fix c , and reverse the roles of left and right in Lemma 6.3. We conclude that, for any $d \geq 7$ and for our setting of μ , G is with high probability a $(1, \mu)$ -right unique neighbor expander. This implies that if $|T| \leq \mu|R|$, then $N^{\text{odd}}(T) \neq \emptyset$ because $N^{\text{odd}}(T) \supseteq N^1(T)$ and $N^1(T) \neq \emptyset$.

Lemma 6.6 says that for any $\mu > 0$, and for our selection of d , all sets of size at least μm have a nonempty odd neighborhood. (Actually, the lemma shows that the odd neighborhood is of linear size, which is more than we need here.)

This completes the proof of the first claim.

- (ii) Notice that if $T \subseteq R$, then $N^{\text{odd}}(T)$ is exactly the support of $\sum_{j \in T} A_j$. Thus, it suffices to show that $N^{\text{odd}}(T)$ is large for large subsets T .

By definition of d, μ, δ and by Lemma 6.6, with high probability G is a $(\delta n, \mu)$ -right odd expander. This means $\mathcal{A}(G)$ is $(\delta n, \mu)$ -local. Part (ii) is proved.

- (iii) Let G_{-j} be the graph obtained from G by removing vertex $j \in R$ and all edges touching it. Since $\mathcal{A}(G)$ is linearly independent, it is sufficient to show that $\mathcal{C}(G_{-j})$ has no nonzero element of Hamming weight $< \varepsilon n$.

Let x be a nonzero element of $\mathcal{C}(G_{-j})$, and let $S_x \subseteq L$ be the set of coordinates at which x is 1. Consider the graph G_{-j} . In this graph, the set of unique neighbors of S_x is empty because $x \in \mathcal{C}(G_{-j})$ (otherwise, some $j' \in N^1(S_x)$, so $\langle A_{j'}, x \rangle = 1$, a contradiction). Thus,

$$(6.4) \quad N^1(S_x) \subseteq \{j\},$$

where $N^1(S_x)$ is the set of unique neighbors of S_x in G . Clearly, $|S_x| > 1$ because the left degree of G is $c > 1$. But if $|S_x| \leq \frac{1}{100} \cdot d^{-2} \cdot n$, then by Lemma 6.3 $|N^1(S_x)| \geq |S_x| > 1$, in contradiction to (6.4). We conclude that for any $x \in \mathcal{C}(G_{-j})$, $|x| \geq \frac{1}{100} \cdot d^{-2}$, so $\mathcal{A}(G)$ is ε -separating for our selection of ε . Part (iii) is complete.

This completes the proof of Lemma 3.6. \square

7. Reducing d LIN to 3LIN. This section proves Lemma 3.8. The randomized construction from section 6 produces d -linear formulae, which are hard to test for some constant d . We would like to make d as small as possible. This section obtains 3-linear, hard-to-test formulae. First, we give a reduction from d -linear to $\lceil \frac{d}{2} \rceil + 1$ -linear formulae and then apply it d times to get 3-linear formulae.

Let φ be a d -linear formula on variables in $X = \{x_1, \dots, x_n\}$. The reduction maps φ to a $(\lceil \frac{d}{2} \rceil + 1)$ -linear formula on variables $X \cup Z$, where Z is a collection of new variables $\{z_1, \dots, z_m\}$. For each constraint A_i , say $x_1 \oplus \dots \oplus x_d = 0$, in φ , two constraints, A_i^1 and A_i^2 , are formed: $x_1 \oplus \dots \oplus x_{\lceil \frac{d}{2} \rceil} \oplus z_i = 0$ and $x_{\lceil \frac{d}{2} \rceil + 1} \oplus \dots \oplus x_d \oplus z_i = 0$. Let $V \subseteq \{0, 1\}^n$ be the vector space of vectors satisfying φ , and let \mathcal{A} be an m -dimensional basis for the vector space V^\perp of constraints. Define $\mathcal{R}(\mathcal{A})$ to be the collection of $2m$ vectors in $\{0, 1\}^{n+m}$ formed by splitting every constraint in \mathcal{A} in two, as described above.

The following three claims show that the reduction preserves the properties which make the formula hard to test. A parameter followed by a prime denotes the value of the parameter after one application of the reduction: for example, $m' = 2m$, $n' = m + n$, and $d' = \lceil \frac{d}{2} \rceil + 1$.

CLAIM 7.1. $\mathcal{R}(\mathcal{A})$ is independent.

Proof. It is enough to prove that no set of constraints in $\mathcal{R}(\mathcal{A})$ sums up to 0. Let C be a subset of constraints in $\mathcal{R}(\mathcal{A})$. If only one of the two constraints involving a new variable z appears in C , then the sum of vectors in C has 1 in z 's position. If, on the other hand, all constraints appear in pairs, then the sum of vectors in C is equal to the sum of the constraints in \mathcal{A} from which C 's constraints were formed. By independence of old constraints, this sum is not 0. \square

CLAIM 7.2. If \mathcal{A} is ε -separating, then $\mathcal{R}(\mathcal{A})$ is ε' -separating, where $\varepsilon' = \frac{\varepsilon}{1 + (m/n)}$.

Proof. Let x' be a vector in $\{0, 1\}^{n+m}$ that falsifies exactly one constraint, say A_i^1 , in $\mathcal{R}(\mathcal{A})$. Namely, $\langle x', A_i^1 \rangle = 1$ and $\langle x', A' \rangle = 0$ for all $A' \in \mathcal{R}(\mathcal{A})$, $A' \neq A_i^1$. Let $x = x'_1 \dots x'_n$. Then $\langle x, A_i \rangle = \langle x', A_i^1 + A_i^2 \rangle = \langle x', A_i^1 \rangle + \langle x', A_i^2 \rangle = 1$, and similarly, $\langle x, A \rangle = 0$ for all $A \in \mathcal{A}$, $A \neq A_i$. Thus, x falsifies exactly one constraint in \mathcal{A} . Since \mathcal{A} is ε -separating, $|x| \geq \varepsilon n$. It follows that $|x'| \geq \varepsilon n$, implying that $\mathcal{R}(\mathcal{A})$ is $(\frac{\varepsilon n}{n+m})$ -separating. \square

CLAIM 7.3. If \mathcal{A} is (q, μ) -local, then $\mathcal{R}(\mathcal{A})$ is (q', μ') -local, where $q' = \frac{q}{d'}$ and $\mu' = \mu + \frac{q'}{m'}$.

Proof. Let $\alpha' \in \{0, 1\}^{m+n}$ be the sum of a subset T of $\mu' \cdot m'$ constraints in $\mathcal{R}(\mathcal{A})$. Let T_2 be the subset of constraints in T which appear in pairs. Namely, for every new variable z , both constraints with z are either in T_2 or not in T_2 . Let $T_1 = T \setminus T_2$.

Case 1. $|T_1| \geq q'$. For every constraint in T_1 , the new variable z from that constraint does not appear in any other constraint in T . Therefore, α' is 1 on z 's coordinate. Hence, $|\alpha'| \geq |T_1| \geq q'$.

Case 2. $|T_1| < q'$. Then $|T_2| = |T| - |T_1| \geq \mu' \cdot m' - q' = \mu \cdot m' = 2\mu m$. Let S be the set of constraints in \mathcal{A} that gave rise to constraints in T_2 . Then $|S| = |T_2|/2 \geq \mu m$. Old variables appear in the same number of constraints in S and in T_2 . Thus,

$$\left| \sum_{A' \in T_2} A' \right| \geq \left| \sum_{A \in S} A \right| \geq q.$$

The last inequality follows from the fact that \mathcal{A} is (q, μ) -local. When constraints from T_1 are added to $\sum_{A' \in T_2} A'$, each T_1 constraint zeros out at most $\lceil \frac{d}{2} \rceil = d' - 1$ coordinates:

$$|\alpha'| \geq \left| \sum_{A' \in T_2} A' \right| - \frac{d}{2} \left| \sum_{A' \in T_1} A' \right| \geq q - (d' - 1)q' = q'. \quad \square$$

If the reduction is applied $\lceil \log(d-2) \rceil$ times, the number of terms in a constraint drops to 3. To see this, think of applying the reduction i times to a formula with $d \leq 2^i + 2$ terms per constraint. Successive iterations will decrease the clause size to $\leq 2^{i-1} + 2, \leq 2^{i-2} + 2$, etc. We apply the reduction $\lceil \log d \rceil$ times to obtain hard 3LIN formulae from hard dLIN formulae, as shown in Lemma 3.8.

Proof of Lemma 3.8. For $\mathcal{A} \subset \{0, 1\}^n$ as in the statement of our lemma, let $\mathcal{R}^{(0)}(\mathcal{A}) = \mathcal{A}$, and for $i \geq 1$ let $\mathcal{R}^{(i)}(\mathcal{A}) = \mathcal{R}(\mathcal{R}^{(i-1)}(\mathcal{A}))$. Let $\mathcal{A}^* = \mathcal{R}^{(\lceil \log d \rceil)}(\mathcal{A})$. As explained above, each constraint in \mathcal{A}^* has weight at most 3. We now calculate the remaining parameters of \mathcal{A}^* . In doing so, we denote the value of a parameter in

$\mathcal{R}^{(i)}(\mathcal{A})$ by the superscript $^{(i)}$, and the superscript * signifies the final value of the parameter in \mathcal{A}^* .

Since each application of the reduction doubles the dimension, $m^* = 2^{\lceil \log d \rceil} m \leq 2dm$. To calculate n^* , observe that the reduction does not change $m - n$ and recall that $dm = cn$. Therefore,

$$n^* = n + m^* - m \leq n + 2dm = (2c + 1) \cdot n.$$

Claim 7.1 guarantees that \mathcal{A}^* is independent. By Claim 7.2, $\varepsilon' = \frac{\varepsilon}{1+(m/n)} = \varepsilon \frac{n}{n^*}$. Thus,

$$\varepsilon \geq \varepsilon^* = \varepsilon \cdot \frac{n}{n^{(1)}} \cdot \frac{n^{(1)}}{n^{(2)}} \cdots \frac{n^{(\lceil \log d \rceil - 1)}}{n^*} = \varepsilon \frac{n}{n^*} \geq \frac{\varepsilon}{2c + 1}.$$

Let $q = \delta n$. Applying Claim 7.3 $\lceil \log d \rceil$ times, we obtain

$$\begin{aligned} q \geq q^* &= \frac{q}{d^{(1)} \times d^{(2)} \times \dots \times d^*} \geq \frac{q}{d^{\lceil \log d \rceil}} \geq \frac{q}{d^{\log d + 1}}; \\ \delta \geq \delta^* &= \frac{q^*}{n^*} \geq \frac{q}{d^{\log d + 1} \cdot (2c + 1)n} = \frac{\delta}{d^{\log d + 1} \cdot (2c + 1)}; \\ \mu^* &= \mu + \frac{q^{(1)}}{m^{(1)}} + \frac{q^{(2)}}{m^{(2)}} + \dots + \frac{q^*}{m^*} \\ &< \mu + \frac{q}{m} \left(\frac{1}{2d^{(1)}} + \frac{1}{4d^{(1)}d^{(2)}} + \dots + \frac{1}{d \cdot d^{(1)}d^{(2)} \dots d^*} \right) \\ &\leq \mu + \frac{q}{m} \frac{\lceil \log d \rceil}{d} \leq \mu + \frac{d\delta n}{cn} \cdot \frac{\log d + 1}{d} = \mu + \frac{\delta(\log d + 1)}{c}. \end{aligned}$$

This completes the proof of Lemma 3.8. \square

Acknowledgments. We thank Madhu Sudan for (i) many helpful conversations, (ii) suggesting the reductions of section 5, and (iii) allowing us to include the “hard to test” properties based on Reed–Muller codes. We thank Piotr Indyk for referring us to Azuma’s inequality to simplify the analysis and thank Michael Sipser for helpful discussions. We are grateful to Oded Goldreich for allowing us to include the “hard to test” properties based on random linear codes. We thank Oded Goldreich and the anonymous referees for several useful suggestions which improved the presentation of the paper.

REFERENCES

- [AFKS00] N. ALON, E. FISCHER, M. KRIVELEVICH, AND M. SZEGEDY, *Efficient testing of large graphs*, *Combinatorica*, 20 (2000), pp. 451–476. (A preliminary version appears in Proceedings of the 40th Annual Symposium on Foundations of Computer Science (FOCS), IEEE, Los Alamitos, CA, 1999, pp. 656–666.)
- [AKNS01] N. ALON, M. KRIVELEVICH, I. NEWMAN, AND M. SZEGEDY, *Regular languages are testable with a constant number of queries*, *SIAM J. Comput.*, 30 (2001), pp. 1842–1862. (A preliminary version appears in Proceedings of the 40th Annual Symposium on Foundations of Computer Science (FOCS), IEEE, Los Alamitos, CA, 1999, pp. 645–655.)
- [AS03] N. ALON AND A. SHAPIRA, *Testing satisfiability*, *J. Algorithms*, 47 (2003), pp. 87–103. (A preliminary version appears in Proceedings of the 13th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA), SIAM, Philadelphia, ACM, New York, 2002, pp. 645–654).

- [BGH⁺04] E. BEN-SASSON, O. GOLDBREICH, P. HARSHA, M. SUDAN, AND S. VADHAN, *Robust PCPs of proximity, shorter PCPs and applications to coding*, in Proceedings of the 36th Annual ACM Symposium on Theory of Computing, Chicago, IL, June 13–15, 2004, pp. 1–10.
- [BHR03] E. BEN-SASSON, P. HARSHA, AND S. RASKHODNIKOVA, *Some 3CNF properties are hard to test*, in Proceedings of the 35th Annual ACM Symposium on Theory of Computing, San Diego, CA, June 9–11, 2003, pp. 345–354.
- [BSVW03] E. BEN-SASSON, M. SUDAN, S. VADHAN, AND A. WIGDERSON, *Randomness-efficient low degree tests and short PCPs via epsilon-biased sets*, in Proceedings of the 35th Annual ACM Symposium on Theory of Computing, San Diego, CA, June 9–11, 2003, pp. 612–621.
- [BLR93] M. BLUM, M. LUBY, AND R. RUBINFELD, *Self-testing/correcting with applications to numerical problems*, J. Comput. System Sci., 47 (1993), pp. 549–595. (A preliminary version appears in Proceedings of the 22nd Annual Symposium on Theory of Computing (STOC), ACM, New York, 1990, pp. 73–83.)
- [BOT02] A. BOGDANOV, K. OBATA, AND L. TREVISAN, *A lower bound for testing 3-colorability in bounded-degree graphs*, in Proceedings of the 43rd Annual IEEE Symposium on Foundations of Computer Science, Vancouver, Canada, Nov. 16–19, 2002, pp. 93–102.
- [CS88] V. CHVÁTAL AND E. SZEMERÉDI, *Many hard examples for resolution*, J. ACM, 35 (1988), pp. 759–768.
- [Fis01] E. FISCHER, *The art of uninformed decisions: A primer to property testing*, Bull. European Assoc. Theoret. Comput. Sci., 75 (2001), pp. 97–126.
- [Fis05] E. FISCHER, *Testing graphs for colorability properties*, Random Structures Algorithms, 26 (2005), pp. 289–309. (A preliminary version appears in Proceeding of the 12th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA), SIAM, Philadelphia, ACM, New York, 2001, pp. 873–882.)
- [FLN⁺02] E. FISCHER, E. LEHMAN, I. NEWMAN, S. RASKHODNIKOVA, R. RUBINFELD, AND A. SAMORODNITSKY, *Monotonicity testing over general poset domains*, in Proceedings of the 34th Annual ACM Symposium on Theory of Computing, New York, May 19–21, 2002, pp. 474–483.
- [FN04] E. FISCHER AND I. NEWMAN, *Functions that have read-twice constant width branching programs are not necessarily testable*, Random Structures Algorithms, 24 (2004), pp. 175–193. (A preliminary version appears in Proceedings of the 17th Annual Conference on Computational Complexity, IEEE, Los Alamitos, CA, 2002, pp. 55–61.)
- [Gal63] R. G. GALLAGER, *Low Density Parity Check Codes*, MIT Press, Cambridge, MA, 1963.
- [GGR98] O. GOLDBREICH, S. GOLDWASSER, AND D. RON, *Property testing and its connection to learning and approximation*, J. ACM, 45 (1998), pp. 653–750. (A preliminary version appears in Proceedings of the 37th Annual Symposium on Foundations of Computer Science (FOCS), IEEE, Los Alamitos, CA, 1996, pp. 339–348.)
- [GR02] O. GOLDBREICH AND D. RON, *Property testing in bounded degree graphs*, Algorithmica, 32 (2002), pp. 302–343. (A preliminary version appears in Proceedings of the 29th Annual Symposium on Theory of Computing (STOC), ACM, New York, 1997, pp. 406–415.)
- [GS02] O. GOLDBREICH AND M. SUDAN, *Locally testable codes and PCPs of almost linear length*, in Proceedings of the 43rd Annual IEEE Symposium on Foundations of Computers Science, Vancouver, Canada, Nov. 16–19, 2002, pp. 13–22.
- [GT03] O. GOLDBREICH AND L. TREVISAN, *Three theorems regarding testing graph properties*, Random Structures Algorithms, 23 (2003), pp. 23–57. (A preliminary version appears in Proceedings of the 42nd Annual Symposium on Foundations of Computer Science (FOCS), IEEE, Los Alamitos, CA, 2001, pp. 460–469.)
- [MR95] R. MOTWANI AND P. RAGHAVAN, *Randomized Algorithms*, Cambridge University Press, Cambridge, UK, 1995.
- [New02] I. NEWMAN, *Testing membership in languages that have small width branching programs*, SIAM J. Comput., 31 (2002), pp. 1557–1570. (A preliminary version appears in Proceedings of the 41st Annual Symposium on Foundations of Computer Science (FOCS), IEEE, Los Alamitos, CA, 2000, pp. 251–258.)
- [Ron01] D. RON, *Property testing (a tutorial)*, in Handbook of Randomized Computing, Comb. Optim. 9, S. Rajasekaran, P. M. Pardalos, J. H. Reif, and J. D. P. Rolim, eds., Kluwer Academic Publishers, Dordrecht 2001, pp. 597–649.

- [RS96] R. RUBINFELD AND M. SUDAN, *Robust characterizations of polynomials with applications to program testing*, SIAM J. Comput., 25 (1996), pp. 252–271. (Preliminary versions appear in Proceedings of the 23rd Symposium on Theory of Computing (STOC), ACM, New York, 1991, pp. 33–42 and Proceedings of the 3rd Annual ACM-SIAM Symposium on Discrete Algorithms (SODA), SIAM, Philadelphia, ACM, New York, 1992, pp. 23–32.)
- [Spi95] DANIEL A. SPIELMAN, *Computationally Efficient Error-Correcting Codes and Holographic Proofs*, Ph.D. thesis, Massachusetts Institute of Technology, Cambridge, MA, 1995.

A PROBABILISTIC ANALYSIS OF TRIE-BASED SORTING OF LARGE COLLECTIONS OF LINE SEGMENTS IN SPATIAL DATABASES*

MICHAEL LINDENBAUM[†], HANAN SAMET[‡], AND GISLI R. HJALTASON[§]

Abstract. The size of five trie-based methods of sorting large collections of line segments in a spatial database is investigated analytically using a random lines image model and geometric probability techniques. The methods are based on sorting the line segments with respect to the space that they occupy. Since the space is two-dimensional, the trie is formed by interleaving the bits corresponding to the binary representation of the x and y coordinates of the underlying space and then testing two bits at each iteration. The result of this formulation yields a class of representations that are referred to as *quadtrie* variants, although they have been traditionally referred to as *quadtree* variants. The analysis differs from prior work in that it uses a detailed explicit model of the image instead of relying on modeling the branching process represented by the tree and leaving the underlying image unspecified. The analysis provides analytic expressions and bounds on the expected size of these quadtree variants. This enables the prediction of storage required by the representations and of the associated performance of algorithms that rely on them. The results are useful in the following two ways:

1. They reveal the properties of the various representations and permit their comparison using analytic, nonexperimental criteria. Some of the results confirm previous analyses (e.g., that the storage requirement of the MX quadtree is proportional to the total lengths of the line segments). An important new result is that for a PMR and Bucket PMR quadtree with sufficiently high values of the splitting threshold (i.e., ≥ 4) the number of nodes is proportional to the number of line segments and is independent of the maximum depth of the tree. This provides a theoretical justification for the good behavior and use of the PMR quadtree, which so far has been only of an empirical nature.
2. The random lines model was found to be general enough to approximate real data in the sense that the properties of the trie representations, when used to store real data (e.g., maps), are similar to their properties when storing random lines data. Therefore, by specifying an equivalent random lines model for a real map, the proposed analytical expressions can be applied to predict the storage required for real data. Specifying the equivalent random lines model requires only an estimate of the effective number of random lines in it. Several such estimates are derived for real images, and the accuracy of the implied predictions is demonstrated on a real collection of maps. The agreement between the predictions and real data suggests that they could serve as the basis of a cost model that can be used by a query optimizer to generate an appropriate query evaluation plan.

Key words. large spatial databases, tries, sorting line segments, geometric probability, analysis of algorithms, spatial data structures, quadtrees, quadtries, cost model, query evaluation

AMS subject classification. 68W40

DOI. 10.1137/S0097539700368527

*Received by the editors February 25, 2000; accepted for publication (in revised form) March 24, 2005; published electronically September 8, 2005. This work was supported in part by the National Science Foundation under grants IRI-9712715, EIA-99-00268, EIA-99-01636, EAR-99-05844, IIS-00-86162, and EIA-00-91474 and by Microsoft Research.

<http://www.siam.org/journals/sicomp/35-1/36852.html>.

[†]Computer Science Department, Technion, 32000 Haifa, Israel (mic@cs.technion.ac.il).

[‡]Computer Science Department, University of Maryland, College Park, MD 10742 (hjs@umiacs.umd.edu).

[§]Deceased. Gisli Hjaltason was a graduate of the University of Maryland and a new faculty member at the University of Waterloo. He passed away tragically and unexpectedly on June 19, 2003. We mourn the loss of this promising scientist and dear friend.

1. Introduction.

1.1. Background. The efficient management of data in large database systems depends on grouping the data in such a way that similar data are aggregated and also stored in proximity so that they can be operated upon at the same time or at approximately the same time (see, e.g., [17]). This grouping is usually achieved by sorting the data. The rationale for sorting the data is to facilitate the presentation of the data to the user (e.g., in reports) and also to speed up query processing using sort-based algorithms such as merge-join.

Although sorting has traditionally been applied to one-dimensional data, it is also applicable to data of higher dimensions. This data can consist of points in a higher dimensional space or of spatial objects that span the higher dimensional space (e.g., lines, regions, surfaces, volumes, etc.). In the case of spatial data in more than one dimension, which is the focus of this paper, the result of applying conventional sorting techniques does not always lead to simpler algorithms. For example, suppose that the data are sorted with respect to a particular reference point (e.g., all U.S. cities, represented as points, are sorted with respect to their distance from Chicago). In this case, if we wish to obtain the points in the order of their distance from another point (e.g., with respect to their distance from Omaha), then the sorting process will, in general, have to be reapplied to the entire set of data. The problem is that the data were sorted in an explicit manner. Instead, we need methods that provide an implicit ordering. Examples of such techniques are called *bucketing methods* (see, e.g., [43, 45]). In this case, the data are sorted on the basis of the space that they occupy and are grouped into cells (i.e., buckets) of a finite capacity.

There are two principal methods for sorting spatial data. The first makes use of an object hierarchy. It is based on propagating the space occupied by groups of the data objects up through the hierarchy (e.g., members of the R-tree family [3, 18]). We do not deal with this method in this paper. The second is based on a decomposition of the space occupied by the data into disjoint cells which are aggregated into larger cells (e.g., members of the quadtree family [44, 43, 45]). The decomposition can be either tree-based or trie-based. The distinction is that the former is applied to the values of the data (e.g., a binary search tree), while the latter makes use of the digits (termed a *trie* [5, 16, 28]) that comprise the domain of the values of the data. Data structures that make use of the latter in one dimension are also known as *digital trees* [28].

Our data consists mainly of line segments in two-dimensional space. Our focus is on using tries to sort the line segments with respect to the space that they occupy. We use tries because they result in partitioning different data sets at the same positions, thereby making it very easy and efficient to use merge-join query processing algorithms. Since the space is two-dimensional, the trie is formed by interleaving the bits corresponding to the binary representation of the x and y coordinates of the underlying space. Two similar, yet still different, trie-based data structures may be created depending on whether we test one bit at each iteration (a k - d *trie* [15]) or two bits at each iteration (a *quadtrie* [21, 43, 45]). In this paper, we focus on *quadtries* for collections of line segments. Unfortunately, the representations that make use of *quadtries* have been traditionally referred to as *quadtree* variants (see, e.g., [26, 44, 43, 45]). In our discussion, all *quadtrees* are based on tries and we precede the term *quadtree* with an appropriate qualifier whenever there is a potential for confusion. Thus the *quadtrees* that we discuss are distinct from those based on multidimensional binary search trees that are used for points (see, e.g., [13, 14]).

Variants of *quadtree* structures have been used for many different spatial objects

including points, regions, lines, rectangles, surfaces, volumes, etc. Algorithms using them generally have good average execution times while maintaining a relatively compact representation [44, 43, 45]. To use these data structures in a database application, we must be able to predict their size. The most obvious advantage of such a capability is that it enables us to determine how much space will be required to store different data sets and to choose between different data structures from an efficiency standpoint. It may also be of use at query evaluation time to aid the estimation of the cost of a particular query execution plan (i.e., processing strategy) to be used by a query optimizer. For example, suppose that we are using a filter-and-refine strategy [6] for processing a window query. In particular, suppose further that we have a method of estimating the number of data structure blocks that intersect the window based on the window's size (see, e.g., [40]). Our results could be used in a reverse sense to estimate the number of objects (i.e., line segments in our case) that intersect these blocks. This could serve as a measure of the cost of the refinement step, which must subsequently determine which of the lines actually intersect the query window. Continuing the filter-and-refine query processing strategy, suppose that we are using the histogram method (see, e.g., [29, 31, 34]) for estimating the number of objects that intersect the query window. We can now plug this information into our results to estimate the number of data structure blocks that intersect the window. This is a good measure of the cost of the filter step, i.e., the I/O (Input/Output) cost for the spatial data structure. An alternative factor in measuring performance when data is disk-based is to examine how the various data structure blocks are declustered on various disks (see, e.g., [33]), but this is beyond the scope of this paper and is not discussed further here.

1.2. Related work. Traditional worst-case analysis is often inappropriate because the worst case tends to be both very bad and highly improbable. Thus most approaches to the analysis of hierarchical data structures have been statistical in nature.

A number of statistical approaches have been tried. The most common uses a uniform distribution in the underlying space (see, e.g., [2, 11]). An alternative is to use a nonuniform distribution. Some techniques that have been used include the Gaussian distribution [36] as well as the clustering of uniformly distributed points [38] or even predetermined shapes [3]. Another approach uses a fractal distribution [9] that has the advantage of exhibiting self-similarity, which means that portions of a part of the data set are statistically similar to the entire data set. The key to this approach is to compute a fractal dimension for a particular point data set and then use it in a query optimizer.

The methods described above are for point data sets.¹ In this paper, we are interested in data where every object has nonzero size (e.g., collections of lines, regions, and so on, instead of being restricted to collections of points). Tamminen [54] considers the performance of quadtrees and bintrees under the assumption that the image consists of a single random line treated as a region, and he analyzes the number of nodes in them using geometric probability. Shaffer, Juvvadi, and Heath [52] follow Tamminen's approach and use a local straight line model to perform an analysis that yields the relative (rather than absolute) storage requirements of the region quadtree and bintree data structures. Other works on region data include Dyer [8], Shaffer [51], and

¹The fractal model has been applied to points derived from a collection of line segments in the sense that the points corresponded to intersections of line segments [9]. This was used to predict the effective occupancy of nodes in an R-tree that stores point data.

Faloutsos, Jagadish, and Manolopoulos [10], Mathieu, Puech, and Yahia [30], Puech and Yahia [42], and Vassilakopoulos and Manolopoulos [56], and these investigate the size of quadtree representations of region data and study some other related questions using some assumptions on the branching probabilities of nodes in the tree. Nelson and Samet [35, 36, 37] consider the distributions of node occupancies in hierarchical geometric data structures that store a variable number of geometric data items per node, which include points and lines. However, the methods of Nelson and Samet have much wider applicability. This approach is similar to hashing [28], where each node acts like a bucket.

Although these approaches sometimes lead to remarkable agreement between theory and simulation (see, e.g., [1, 36]), they have a common drawback. The explicit model of the image on which the statistical analysis is performed is either exceedingly simple or is not given at all. When the model is not given, the model must be implied from other assumptions. In particular, an assumption is made on the splitting probability in the data structure (see, e.g., [1, 30, 36, 42]), which implies the existence of some *implicit* model on the data. However, when we ask what kind of data (real or contrived) fit this model, the only possible answer is a circular one that says that the data give rise to these probabilities. Unfortunately, there is no explicit indication of whether there exists some image model associated with these splitting probabilities. Thus the connection between the analysis and the performance with real image data is not clear. In contrast, our approach, as described below, is to use an *explicit* nontrivial random image model and to then show that data can be generated corresponding to this model, which also fits the analysis. Note that we are not claiming that the data we generate correspond exactly to typical images, although we deal with this issue as well. In this sense our approach is complementary to the work of Flajolet and Puech [15], who analyzed the partial match query time for hierarchical data structures, while we analyze their storage requirements. Unlike their data, which consist of random points in a high-dimensional space whose coordinate values are drawn from a uniform distribution, our data consist of randomly drawn lines. It is important to note that a line is a qualitatively different data type than a point, as the action of every line on the structure is not local.

An alternative nonstatistical approach was applied by Hunter [22] and Steiglitz [23] to show that, for a polygon of perimeter l , the size (i.e., the number of nodes) of the corresponding MX quadtree (a variant of the region quadtree described in more detail in section 2) is $O(l)$. This classic result, although derived for simple polygons, has been observed to be sufficiently general to be useful for predicting the performance of a number of algorithms for different images represented by a region quadtree [48].

As we pointed out above, in this paper we investigate the use of a random image model consisting of M randomly drawn lines. Unlike Tamminen’s approach [54], which considers a single random line, here we treat the much more general and complicated situation of an arbitrary number of lines. We use geometric probability to analyze four variants of the quadtree that can be built for data that obey this model by determining the expected number of nodes in each variant. These variants are the MX quadtree [23, 43, 45], the PM quadtree [49], the PMR quadtree [35, 36, 37], and a new variant of the PMR quadtree representation, which we call a Bucket PMR quadtree (see also [19, 20]).

1.3. Contributions. The analysis that we provide is important for two reasons. First, it allows for a meaningful, quantitative, and analytic comparison of a number of

different options for representing linear spatial data, and it provides tools for choosing between these options in a way which is neither experimental nor domain dependent. The second reason is even more practical: we found that we could actually predict the storage requirements of these representation options by specifying an equivalent random lines data set, with some equivalent number of lines, and use the proposed analytic expressions for predicting its size.

In particular, our analysis shows that for images of the same complexity, the PMR quadtree and the Bucket PMR quadtree for sufficiently high values of the splitting threshold (i.e., ≥ 4) are the most efficient in the sense that they require the least storage. The PM quadtree follows, and the MX quadtree requires the largest amount of storage. This qualitative ordering verifies experimental results obtained in the past [22, 23, 35, 49] and agrees with the extensive experimentation that we have carried out. This verification, along with its accompanying theoretical justification, is one of the contributions of our research.

The agreement between the results of our analysis and the data was not surprising in the case of the MX quadtree because it confirmed previous results (i.e., [22, 23]). However, in the case of the PMR and Bucket PMR quadtrees for sufficiently high values of the splitting threshold (i.e., ≥ 4) our analysis breaks new ground because we are able to derive theoretically and verify experimentally for both random data and real map data that the number of nodes is asymptotically proportional to the number of line segments. This is quite significant, as it enables us to predict the number of nodes required by this representation, and, most importantly, to show that it is independent of the maximum depth of the tree.

It is important to note that we do not claim that our proposed random image model yields data instances which are visually similar to what appears in realistic geometric applications, such as road networks. Nevertheless, we do show that the analysis can be interpreted in terms of the geometric properties of the image, such as line length and the number of intersections between lines. With this interpretation, the predictions, derived for random images, may be applied to real data by measuring the relevant geometric property and using it to specify equivalent random images. Testing the predictions on a real set of maps yielded relatively accurate predictions of the storage required for the maps.

Although our analysis is for a particular data type (i.e., collections of line segments) and data structures, we believe that it has wider applicability. In particular, the geometric probability approach could be extended for data types other than line segments (e.g., points, polygons, surfaces, solids, etc.). Furthermore, the random image model can be used in a statistical analysis of other trie-based spatial data structures.

The rest of this paper is organized as follows. Section 2 gives a brief overview of the quadtree representations of collections of line segments, including the definitions of the four variants that we analyze. Section 3 presents the random image model and reviews some necessary results from geometric probability. Section 4 contains a statistical analysis using the model and the results of its application to each of the aforementioned quadtree variants. It also contains the results of some experiments with instances of the random image model. Section 5 describes the application of the analysis to predict the storage requirements for real data and presents results of extensive experiments that support its validity. Section 6 contains concluding remarks and gives some directions for future research.

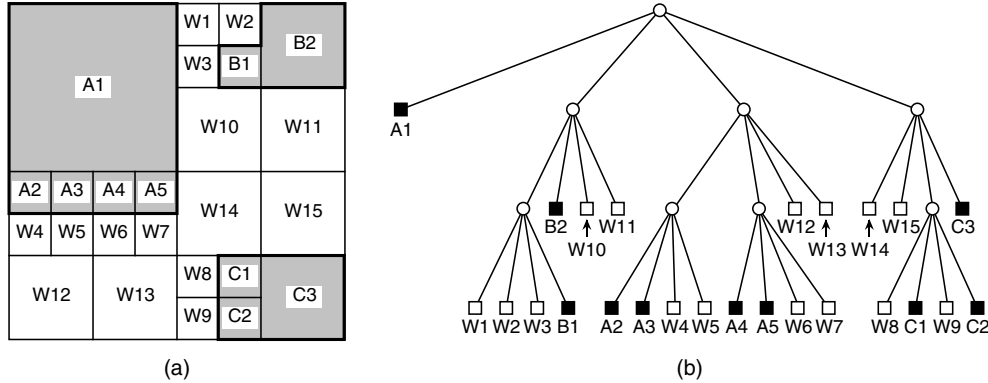


FIG. 1. (a) *Block decomposition* and (b) *its tree representation for the region quadtree corresponding to a collection of three regions A, B, and C.*

2. Overview of quadtree representations of collections of line segments.

A quadtree is a hierarchical variable resolution data structure based on the recursive partitioning of the two-dimensional plane into quadrants. It can be viewed as a 4-ary tree, where each node represents a region in the plane called a block, and the children of each node represent a partition of that region into four parts. This scheme is useful for representing geometric data at a variable resolution. The most commonly known version of the quadtree is the *region quadtree* [26]. It is used for the representation of planar regions. In this case, for two-dimensional data, the environment containing the regions is recursively decomposed into four rectangular congruent blocks until each block either is completely occupied by a region or is empty (such a decomposition process is termed *regular*). For example, Figure 1(a) is the block decomposition for the region quadtree corresponding to three regions A, B, and C. Notice that, in this case, all the blocks are square, have sides whose size is a power of 2, and are located at specific positions.

The traditional, and most natural, access structure for a region quadtree corresponding to a two-dimensional image is a tree with a fanout of 4 (see, e.g., Figure 1(b)). Each leaf node in the tree corresponds to a different block b and contains the identity of the region associated with b . Each nonleaf node f corresponds to a block whose volume is the union of the blocks corresponding to the four children of f . In this case, the tree is a containment hierarchy and closely parallels the decomposition in that they are both recursive processes and the blocks corresponding to nodes at different depths of the tree are similar in shape.

Quadtree variants exist for representing other spatial data types than just planar regions. For example, they are used to represent collections of points [12, 46], collections of line segments [35, 36, 37, 49], as well as more complicated objects (e.g., rectangles [25]). Generalizations of the quadtree to three and higher dimensions (e.g., octrees [22, 24, 32] and bintrees [27, 48, 55]) have also been investigated. These generalizations have many of the same basic properties.

The different variants of the quadtree data structure can be subdivided into two categories: those based on a regular decomposition of space using predefined positions for the partition lines (i.e., trie-based), and those in which the positions of the partition lines are determined explicitly by the data as they are inserted into the data structure (i.e., tree-based or data-based). In most cases, use of regular decomposition indicates

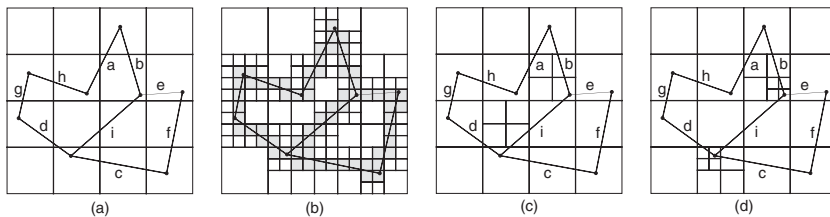


FIG. 2. (a) Collection of line segments in a 4×4 grid, (b) its MX quadtree, (c) its PM quadtree, and (d) its Bucket PMR quadtree with a bucket capacity of 2. Only the resulting decomposition of the underlying space into blocks is shown. The corresponding tree structure that acts as an access structure to ensure logarithmic access times is not shown.

that the shape of the resulting data structure is independent of the order in which the data are inserted into the structure when building it. This is not the case for data-based decompositions. Interestingly, for most applications, regular decomposition works at least as well as data-based decomposition. Moreover, regular decomposition is easier to implement and analyze. At times, a distinction between different variants of the quadtree data structure is also made on the basis of whether or not there is a predefined maximum depth (denoted by N).

In this paper we only consider quadtree representations of collections of line segments. We restrict ourselves to quadtrees based on a regular decomposition. In the rest of this section we review the different quadtree variants that we study. They differ in the condition that is used to determine when a quadtree block should be decomposed—this condition is termed a *splitting rule*.

The simplest quadtree representation is the *MX quadtree*, which assumes that the underlying domain of the data is a $2^N \times 2^N$ grid. The MX quadtree is built by digitizing the line segments and labeling each unit-sized cell (i.e., pixel) through which it passes as being of type **boundary**. The remaining pixels are marked **WHITE** and are merged, if possible, into larger and larger quadtree blocks, as done in the region quadtree. Figure 2(b) is the MX quadtree for the collection of line segment objects in Figure 2(a). A drawback of the MX quadtree is that it associates a thickness with a line. Also, it is difficult to detect the presence of a vertex whenever five or more line segments meet. The above definition of the MX quadtree is given in a bottom-up manner. We can also define it in a top-down manner. In particular, we start with one block corresponding to the entire $2^N \times 2^N$ space and recursively decompose it into four blocks, halting the decomposition when a block is empty or is of size 1×1 (i.e., the block corresponds to a pixel).

The PM quadtree is a refinement of the MX quadtree that is motivated by the observation that the number of blocks in the decomposition can be reduced by terminating the subdivision whenever a line segment passes through a block completely (i.e., it enters and exits the block rather than starting or terminating in the block). Nevertheless, even when this observation is used, the resulting structure still has full decomposition at each vertex or endpoint of a line segment. To avoid this situation, we further modify the above top-down MX quadtree definition so that decomposition takes place as long as more than one line segment passes through the block, unless all of the line segments that pass through the block are incident at the same vertex, which is also required to be in the same block. In addition, the decomposition is also halted whenever a 1×1 block is encountered. The PM quadtree is the structure that results when these additional halting conditions are imposed on the top-down definition of

the MX quadtree. The fact that the PM quadtree is defined in a top-down manner means that each block is maximal. It should be clear that each block at a depth less than the maximum depth contains at most one vertex. For example, Figure 2(c) is the PM quadtree corresponding to the collection of line segment objects in Figure 2(a).

The PM quadtree is vertex-based in the sense that the vertices play a role in determining when the decomposition process stops. In particular, the decomposition process halts when there is more than one line segment in a block, provided that all of the line segments are incident at the same vertex in the block. An alternative is to use an edge-based representation, where a block is split whenever there are more than q line segments passing through it. Thus the blocks serve as buckets with a capacity q . This is known as a *Bucket PMR quadtree*. The drawback of this method is that, whenever more than q line segments are incident at a vertex v , the block containing v will be decomposed until reaching the maximum depth N , which corresponds to a 1×1 block. For example, Figure 2(d) is the Bucket PMR quadtree corresponding to the collection of line segment objects in Figure 2(a) when using a bucket capacity of $q = 2$ and a maximum depth $N = 4$.

As pointed out above, the drawback of the Bucket PMR quadtree is that if the number of line segments incident at a vertex exceeds the bucket capacity, then the decomposition in the neighborhood of the vertex will not halt unless we reach the maximum allowable depth N . This problem is resolved by the *PMR quadtree*, which is similar to a Bucket PMR quadtree with the difference that, in the PMR quadtree, a block is decomposed once, and only once, if the insertion causes it to have more than q line segments. Therefore, in the PMR quadtree, q serves as a *splitting threshold*, which is quite different than a bucket capacity, which is its role in the Bucket PMR quadtree. There is no maximum depth in the PMR quadtree.

The PMR quadtree is constructed by inserting the line segments one by one into an initially empty structure consisting of one block. Each line segment is inserted into all the blocks that it intersects or occupies in its entirety. During this process, the occupancy of each block that is intersected by the line segment is checked to see if the insertion causes it to exceed the splitting threshold. If the splitting threshold is exceeded, the block is split *once*, and only once, into four blocks of equal size.

Figure 3(e) is the PMR quadtree for the collection of line segment objects in Figure 2(a) with a splitting threshold of $q = 2$. The nine line segments, labeled a–i, are inserted in alphabetic order. It should be clear that the shape of the PMR quadtree for a given collection of line segments is not unique; instead, it depends on the order in which the line segments are inserted into it. In contrast, the shapes of the MX, PM, and Bucket PMR quadtrees are unique. Figure 3(a)–(e) shows some of the steps in the process of building the PMR quadtree of Figure 3(e) with a splitting threshold of 2. In each part of Figure 3(a)–(e), the line segment that caused the subdivision is denoted by a thick line, while the gray regions indicate the blocks where a subdivision has taken place.

The insertion of line segments c, e, g, h, and i causes the subdivisions in parts (a), (b), (c), (d), and (e), respectively, of Figure 3. The insertion of line segment i causes three blocks to be subdivided (i.e., the SE block in the SW quadrant, the SE quadrant, and the SW block in the NE quadrant). The final result is shown in Figure 3(e). Note the difference from the PM quadtree in Figure 2(c)—that is, the NE block of the SW quadrant is decomposed in the PM quadtree, while the SE block of the SW quadrant is not decomposed in the PM quadtree. We also observe that, unlike the Bucket PMR quadtree in Figure 2(d), we did not have to split to the maximum

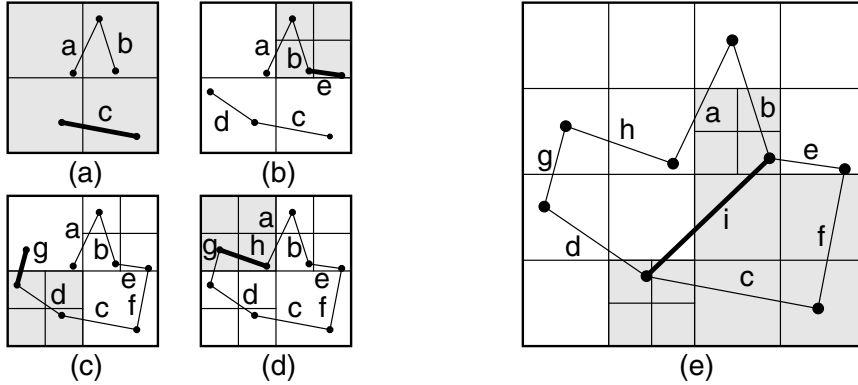


FIG. 3. *PMR quadtree with a splitting threshold of 2 for the collection of line segments of Figure 2(a). (a)–(e) illustrate snapshots of the construction process with the final PMR quadtree given in (e).*

depth in the neighborhood of the vertices, where line segments c , d , i , as well as b , e , i , meet.

It should be clear that the number of line segments in a PMR quadtree block can exceed the value of the splitting threshold q . A value of four for q is usually sufficient to store collections of line segments efficiently, as it implies that junctions of two, three, and four line segments (which are common in maps of roads and rivers, etc.) do not cause a split. Of course, there are situations in which more than four line segments will meet at a vertex. However, we assume that such situations are rare. Note that, at times, we want to express the dependence of the Bucket PMR quadtree and the PMR quadtree on q explicitly, in which case we use the term *bucket PMR_q quadtree* and *PMR_q quadtree*, respectively, to describe the structure.

Note that, in the general case, we may have a collection of line segments that intersect, while the intersection point is not a vertex. Such a situation can result when we are representing a nonplanar graph. As we shall see later, this is not an issue for the line segment arrangements that are generated by our random image model (described in section 3).

3. Random image models. In the first part of this paper (sections 3 and 4) we assumed that the quadtree variants that are discussed represent geometric structures, which are instances of a random process described as follows. Observe that the line $L(\rho, \theta)$ consists of the points (x, y) satisfying the relation

$$L(\rho, \theta) = \{(x, y) \mid x \cos \theta + y \sin \theta = \rho\}.$$

The line $L(\rho, \theta)$ is perpendicular to the vector $(\cos \theta, \sin \theta)$ (see Figure 4(a)). Although the position of every particular line, $L(\rho, \theta)$, naturally depends on the origin and orientation of the coordinate system, we shall soon see that the probability of every random line, drawn according to our model, does not. Therefore, the origin and orientation of the coordinate system relative to the image does not make a difference. The arbitrarily chosen location of the coordinate system in Figure 4(a) illustrates this invariance property. For the rectangular region R ,

$$R = \{(x, y) \mid |x|, |y| < 2^{N-1}\},$$

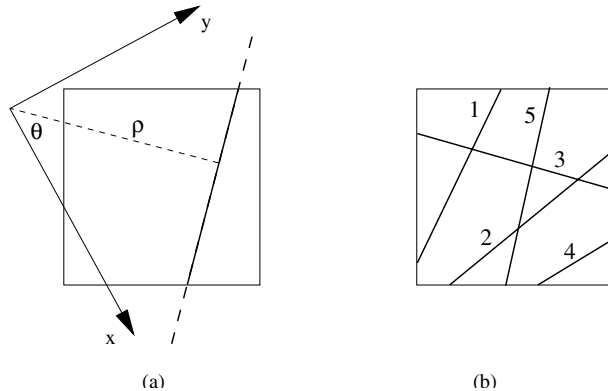


FIG. 4. *The random image model: (a) The process of generating a random line (note that the coordinate system is arbitrary with respect to the image). (b) A typical instance of the random image constructed for $M = 5$ independently drawn lines labeled 1–5.*

let T be the parameter set

$$T = \{(\rho, \theta) | L(\rho, \theta) \cap R \neq \emptyset\},$$

which includes all the parameter pairs (ρ, θ) that represent lines intersecting with R . Let

$$(1) \quad p(\rho, \theta) = \begin{cases} \frac{1}{|T|} & (\rho, \theta) \in T, \\ 0 & \text{otherwise} \end{cases}$$

be a probability density function, where

$$|T| = \int_T d\rho d\theta.$$

This distribution, called the *uniform density distribution*, is the only one which ensures that the probability of choosing a particular random line is independent of the coordinate system in which ρ and θ are defined (i.e., it is independent of the translation or rotation of the coordinate system [50]). Therefore, it is the natural density function to specify when modeling collections of random lines. As an illustration, see Figure 4(b), where an instance of the random image containing five lines labeled 1–5 is described.

Every instance of our random image model is a $2^N \times 2^N$ image with M random lines that intersect it and which are chosen independently according to the density function (1). The continuous uniform density distribution implies that three lines intersect at the same point with probability zero. This follows from the observation that two intersecting lines define a point, say (x_0, y_0) , which can be regarded as prespecified for the third line. The parameters of every line which intersects this point must be in the set $\{(\rho, \theta) | x_0 \cos \theta + y_0 \sin \theta = \rho\}$, the measure of which is zero (i.e., a one-dimensional quantity) with respect to the measure of T (i.e., a two-dimensional quantity). Therefore, after drawing a finite number of random lines, the probability that any three of them will intersect (x_0, y_0) is zero. This property is usually satisfied for real spatial data such as road maps, as intersections of four or more line segments are rare (i.e., junctions of four or more roads).

Each of the line segments clipped from a random infinite line by the boundary of the image is subdivided further into smaller line segments by its intersection with other infinite lines so that, eventually, no line segment crosses another line segment except at the endpoints of the line segments.

Thus the data represented by the various quadtrees (in the first, analytical, part of the paper) are a collection of random line segments specified as a set of M random infinite lines. Note that the infinite lines are not of interest by themselves but are useful for creating the distribution of line segments which we analyze.

We do not claim that our proposed random image model will enable us to generate data that correlate with what appears in realistic geometric applications, such as road networks. Finding such a correlation is unlikely, as realistic geometric data do not consist of lines whose endpoints lie on the image boundary. Nevertheless, the analysis can be interpreted, as we shall see later, in terms of the geometric properties of the image, such as line length and the number of intersections between lines. This allows us to apply its results to real data with similar geometric properties. Most important, the analysis provides us a means to justify claims about the relative qualitative behavior of the different data structures.

Before starting the analysis, we first present three results from geometric probability which form the basis of our results [50].

Geometric probability theorem 1 (Theorem GP1). Let C_1 be a convex planar set included in the convex planar set $C \subset R$. Let L_1 and L be the perimeters of C_1 and C , respectively. Let l be a random line chosen using the uniform density distribution given by (1). Therefore, the probability that a line l passing through C also passes through C_1 is

$$p\{l \cap C_1 \neq \emptyset | l \cap C \neq \emptyset\} = \frac{L_1}{L}.$$

Geometric probability theorem 2 (Theorem GP2). Let $C \subset R$ be a convex planar set with area A and perimeter L . Let l be a random line chosen using the uniform density distribution given by (1). Suppose that l intersects with C and creates a chord H with length $|H|$. Then the expected length of H is

$$E[|H|] = \frac{\pi A}{L}.$$

Geometric probability theorem 3 (Theorem GP3). Let $C \subset R$ be a convex planar set with area A and perimeter L . Let l_1 and l_2 be two random lines, independently chosen using the uniform density distribution (1). If both lines l_1 and l_2 intersect with C , then the probability that l_1 intersects with l_2 inside C is

$$p\{(l_1 \cap l_2) \cap C \neq \emptyset | l_1 \cap C \neq \emptyset, l_2 \cap C \neq \emptyset\} = \frac{2\pi A}{L^2}.$$

4. Statistical analysis of quadtree representations of collections of line segments. An image as defined in section 3 is an instance of a random event. It follows that its hierarchical representation, using one of the quadtree variants, is also a random event. Moreover, the existence of a node in the tree, or its being a leaf, is a random event. Let v be a node of the tree, at depth d , corresponding to some particular cell. Both the existence of v in the tree and its potential split are random events corresponding to the particular arrangement of random lines. Let P_d be the probability that both of these events happen. Recall that the distribution of the

random lines is independent of the coordinate system translation, thereby implying that P_d depends only on the depth.

Let v_e denote the event that the node v exists. Let v_s denote the event that the splitting condition holds for the block corresponding to node v . Letting $\text{Prob}(v_e, v_s)$ be the probability of the joint event that both node v at depth d exists (v_e) and that the splitting condition is satisfied for the square region corresponding to node v (v_s), we have that

$$(2) \quad P_d = \text{Prob}(v_e, v_s) = \text{Prob}(v_s)\text{Prob}(v_e|v_s).$$

$\text{Prob}(v_s)$ is the probability that the splitting condition is satisfied for a particular square region corresponding to a node v (at depth d), while $\text{Prob}(v_e|v_s)$ is the conditional probability that node v at depth d exists, given that the splitting condition holds for the square region corresponding to v . Note that both v_s and v_e depend on the depth d as well.² For example, in the MX quadtree, $\text{Prob}(v_s)$ is the probability that at least one line passes through this region. The node v exists if every member in the sequence of its recursive parents splits as well. For the MX quadtree, this always happens (if v_s is true) because there is at least one line passing through all of them: the line which passes through v and satisfies its splitting condition. Therefore, for the MX quadtree, $\text{Prob}(v_e|v_s) = 1$ and $P_d = \text{Prob}(v_s)$. We shall see later that this relation is not necessarily satisfied for every tree structure (e.g., the PM quadtree as discussed in subsection 4.2.1).

Let S be the total number of nodes in the tree. Every nonleaf node at depth $d-1$ contributes 4 nodes at depth d . The maximal number of nodes at depth $d-1$ is 4^{d-1} , implying that the expected size of the tree is

$$(3) \quad E[S] = 1 + \sum_{d=1}^N 4^d \cdot P_{d-1}.$$

Note that while the splitting events associated with, say, neighboring nodes, are definitely dependent events, this does not effect the calculation of the expected value [41].

Equation (3), which gives the expected number of nodes in the tree, serves as the basis for our analysis. In the following subsections we focus on splitting rules for each of the quadtree variants discussed in section 2. For each rule, we derive the corresponding splitting probabilities P_d and then use (3) to calculate the expected size of the data structure.

4.1. MX quadtree. An MX quadtree represents a collection of line segments on the plane by partitioning the plane into square blocks using the splitting rule that says a block is split if both the depth of its corresponding node is less than N and if the block contains at least one line segment. If the block does not contain a line segment, then it is not subdivided further and its corresponding node is a leaf. Otherwise, it is subdivided and its corresponding node has four children (see Figure 2(b)).

²This “backward” decomposition was preferred over the “more natural” $\text{Prob}(v_e)\text{Prob}(v_s|v_e)$ because it isolates the event v_s , which is “local” and depends only on the configuration of the lines intersecting the block corresponding to v . In contrast, v_e is a more complex event, depending on the existence of all of the ancestors of v .

4.1.1. Analysis. In order to compute the probabilities P_d , we use the following argument. A node (denoted v) at depth d corresponds to a $2^{N-d} \times 2^{N-d}$ square (denoted v as well). Theorem GP1 implies that a particular random line passes through this region with probability

$$p_d = \frac{4 \cdot 2^{N-d}}{4 \cdot 2^N} = \left(\frac{1}{2}\right)^d.$$

The probability that exactly k out of M lines pass through this region is

$$(4) \quad p_{d,k} = \binom{M}{k} \left(\frac{1}{2}\right)^{dk} \left[1 - \left(\frac{1}{2}\right)^d\right]^{M-k}.$$

The probability that this region corresponds to a nonleaf node is

$$(5) \quad P_d = \text{Prob}(v_s)\text{Prob}(v_e|v_s) = \text{Prob}(v_s) = 1 - p_{d,0} = 1 - \left[1 - \left(\frac{1}{2}\right)^d\right]^M,$$

where $\text{Prob}(v_s)$ is the probability that one or more lines pass through this region, thereby satisfying the splitting condition. As mentioned above, for the MX quadtree, $\text{Prob}(v_e|v_s)$ is always 1 because there is at least one line passing through all the regions corresponding to the recursive parents of this region: the line which passes through v and satisfies its splitting condition.

Inserting (5) into (3), we get

$$(6) \quad E[S] = 1 + \sum_{d=1}^N 4^d \cdot P_{d-1} = \sum_{d=1}^N 4^d \left[1 - \left[1 - \left(\frac{1}{2}\right)^{d-1}\right]^M\right].$$

It is difficult to derive closed forms of sums of this nature. To our knowledge, no relevant solutions exist in the literature. Furthermore, we tried, without success, to evaluate it using various symbolic equation solvers, and consulted their developers as well. Therefore, as we are primarily interested in comparing the asymptotic behavior of the storage requirements of the various data structures, we resort to closed form upper and lower bounds for $E[S]$ —that is, the expected number of nodes in the tree. However, for practical use of this estimate, we suggest inserting the known parameters (i.e., M and N) into the sum (6) and evaluating it numerically. These comments are also applicable in the analyses of the rest of the data structures (see subsections 4.2.1 and 4.3).³

Our technique is based on decomposing (6) into two sums \sum_1 and \sum_2 corresponding to the number of nodes at depth less than or equal to d_0 and all the nodes at a depth greater than d_0 , respectively. In essence, our analysis focuses on evaluating the second sum, while the first sum is bounded by the number of nodes in the complete tree (when calculating the upper bound) or by zero (when calculating the lower bound). We find it convenient to formulate our analysis of $E[S]$ in terms of an

³Note that, although the form of the sum (6) intuitively calls for the use of the commonly known $1 + x \leq e^x$ inequality, it does not help here. The problem is that, in the case at hand, we want to bound (6) from above, which means that the term $\left[1 - \left(\frac{1}{2}\right)^{d-1}\right]^M$ should be bound from below, but this is not possible with this inequality.

additional parameter β (as well as M and N), which is defined as

$$(7) \quad \beta = M \left(\frac{1}{2} \right)^{d_0}.$$

The depth d_0 is chosen so that β is less than 1. This enables us to make some assumptions leading to crucial simplifications (i.e., that certain sums converge as the index of summation gets infinitely large). The result (i.e., $E[S]$) is in terms of M , N , and β . Once the result has been obtained, the value of d_0 is adjusted so that the bounds on $E[S]$ are as tight as possible under the constraints that d_0 is an integer bounded by N and that $\beta < 1$.

Decomposing $E[S]$ into two sums \sum_1 and \sum_2 yields

$$(8) \quad E[S] = 1 + \sum_{d=1}^{d_0} 4^d \left[1 - \left[1 - \left(\frac{1}{2} \right)^{d-1} \right]^M \right] + \sum_{d=d_0+1}^N 4^d \left[1 - \left[1 - \left(\frac{1}{2} \right)^{d-1} \right]^M \right] \\ = \sum_1 + \sum_2.$$

$$(9) \quad \sum_1 = 1 + \sum_{d=1}^{d_0} 4^d \left[1 - \left[1 - \left(\frac{1}{2} \right)^{d-1} \right]^M \right] \leq \sum_{d=0}^{d_0} 4^d \cdot 1 \approx \frac{4^{d_0}}{1 - \frac{1}{4}} \approx \frac{4}{3} \frac{M^2}{\beta^2}.$$

Note that what we have done is decompose $E[S]$ into two parts, \sum_1 and \sum_2 , where \sum_1 corresponds to a complete tree at a depth less than or equal to d_0 . Taking the binomial expansion of \sum_2 , we get

$$(10) \quad \sum_2 = \sum_{d=d_0+1}^N 4^d \left[1 - \sum_{k=0}^M \left(\frac{1}{2} \right)^{kd-k} \binom{M}{k} (-1)^k \right] \\ = \sum_{d=d_0+1}^N \sum_{k=1}^M 2^{2d} \left(\frac{1}{2} \right)^{kd-k} \binom{M}{k} (-1)^{k-1}.$$

Changing the order of summation and separating the $k = 1$ (\sum_3), $k = 2$ (\sum_4), and $k > 2$ (\sum_5) cases, we get

$$(11) \quad \sum_2 = \sum_3 + \sum_4 + \sum_5,$$

$$(12) \quad \sum_3 = \sum_{d=d_0+1}^N 2^{2d} \left(\frac{1}{2} \right)^{d-1} \binom{M}{1} \\ = \sum_{d=d_0+1}^N 2^d \cdot 2 \cdot M = 4M(2^N - 2^{d_0}) = 4M \cdot 2^N - \frac{4M^2}{\beta},$$

$$(13) \quad \sum_4 = - \sum_{d=d_0+1}^N 2^{2d} \left(\frac{1}{2} \right)^{2d-2} \binom{M}{2} = - \sum_{d=d_0+1}^N 4 \binom{M}{2} = -2M(M-1)(N-d_0),$$

(14)

$$\begin{aligned}
\sum_5 &= \sum_{d=d_0+1}^N \sum_{k=3}^M 2^k \binom{M}{k} \left(\frac{1}{2}\right)^{(k-2)d} (-1)^{k-1} \\
&\leq \sum_{k=3}^M \sum_{d=d_0+1}^N 2^k \binom{M}{k} \left(\frac{1}{2}\right)^{(k-2)d} \\
&\approx \sum_{k=3}^M 2^k \binom{M}{k} \frac{\left(\frac{1}{2}\right)^{d_0(k-2)} \left(\frac{1}{2}\right)^{k-2}}{1 - \left(\frac{1}{2}\right)^{k-2}} \\
&= \sum_{k=3}^M 4 \binom{M}{k} \left[\frac{\beta}{M}\right]^{k-2} \frac{1}{1 - \left(\frac{1}{2}\right)^{k-2}} \\
&= \sum_{k=3}^M 4 \cdot \frac{M \cdot (M-1) \cdot (M-2) \cdots (M-k+1)}{k!} \frac{\beta^{k-2}}{M^{k-2}} \frac{1}{1 - \left(\frac{1}{2}\right)^{k-2}} \leq \frac{4}{3} M^2 \frac{\beta}{1-\beta}.
\end{aligned}$$

Note that all approximations performed while calculating \sum_1 and \sum_5 are also upper bounds. We continue by summing all contributions, which are partly expected values and partly upper bounds for expected values, to get

(15)

$$\begin{aligned}
E[S] &= \sum_1 + \sum_3 + \sum_4 + \sum_5 \\
&\leq 4 \cdot M \cdot 2^N - 2 \cdot M(M-1) \cdot N + M^2 \left[-\frac{4}{\beta} + \frac{4}{3} \frac{1}{\beta^2} + \frac{4}{3} \frac{\beta}{1-\beta} + 2 \log_2 \frac{M}{\beta} \right].
\end{aligned}$$

Figure 5 shows the value of the sum estimate given by (6) (second curve from the top) as well as the upper bound (upper curve) given by (15) as a function of M at a maximal depth of $N = 10$. Recall that the upper bounds in the figure are minimal in the sense that, for each value of M , upper bounds were calculated for every possible value of d_0 (subject to the constraint $\beta < 1$) and the minimal (tightest) upper bound was taken.

4.1.2. Interpretation. Asymptotically, the dominant contribution to the number of nodes comes from the first term in (15), which may be transformed into a more familiar form using the GP2. Letting L_i be the length of the i th line in our geometric structure, the expected total length L of all lines is

$$(16) \quad E[L] = \sum_{i=1}^M E[L_i] = M \cdot \pi \frac{(2^N)^2}{4 \cdot 2^N} = \frac{\pi}{4} \cdot M \cdot 2^N.$$

Substituting (16) into the first term of (15), we get

$$(17) \quad E[S] \approx \frac{16}{\pi} E[L].$$

In other words, the expected number of nodes in an MX quadtree is proportional to the total expected length of the lines, which agrees with results derived previously under different (nonprobabilistic) models [22, 23].

4.1.3. A lower bound. The derivation of $E[S]$ given by (9)–(15) may be used to set a lower bound on the expected number of nodes. $E[S]$ consists of the contributions

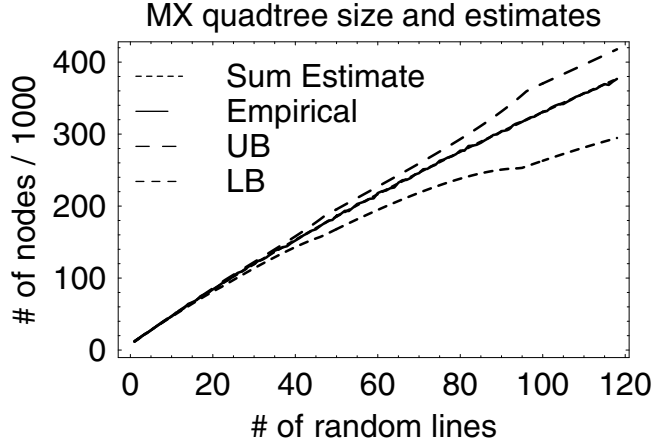


FIG. 5. The upper bound (UB), the model prediction (Sum Estimate), the empirical estimate (Empirical), and the lower bound (LB) on the number of nodes necessary to store an MX quadtree. The empirical estimate almost coincides with the model prediction. These are the two middle curves, which actually look like one curve. The x and y axes correspond to the number of lines and nodes, respectively, for a tree of depth $N = 10$.

of \sum_1 , \sum_3 , \sum_4 , and \sum_5 . \sum_3 and \sum_4 are exact values, \sum_1 is positive, thereby having a lower bound of 0, while \sum_5 can be easily bounded from below by $-\frac{4}{3}M^2\frac{\beta}{1-\beta}$. Thus,

$$(18) \quad E[S] \geq \sum_3 + \sum_4 - \frac{4}{3}M^2\frac{\beta}{1-\beta}.$$

Furthermore, for large N satisfying $2^N \geq M \cdot N$, we have that \sum_1 , \sum_4 , and \sum_5 are small with respect to \sum_3 . Thus the difference between the upper bound and the lower bound is small, and each of the bounds is a good approximation of $E[S]$ (see the lowest curve in Figure 5).

4.2. PM quadtree. Our variant of a PM quadtree represents a collection of line segments in the plane. It partitions the plane into square blocks using the splitting rule that says a block is split unless the depth of the corresponding node is N , or only one line passes through the block, or there is just one vertex in the block and all the lines that pass through the block meet at that vertex. Thus if the block contains a single line, or all the lines pass through a common point in the block and there is no other endpoint in the block, then the block is not subdivided further and its corresponding node is a leaf. Otherwise, it is subdivided and its corresponding node has four children (see Figure 2(c)).

4.2.1. Analysis. Recall from the opening remarks in section 4 that the probability that a region corresponds to a nonleaf node is $P_d = \text{Prob}(v_s)\text{Prob}(v_e|v_s)$. Consider first $\text{Prob}(v_s)$, the probability that the splitting condition is satisfied. Let α be the probability that two lines intersect inside a square region Q given that each of these lines passes through Q . For a square $2^{N-d} \times 2^{N-d}$ region ($0 \leq d \leq N$), the probability that the splitting conditions of a PM quadtree are satisfied for a node v in depth d may be written as

$$(19) \quad \text{Prob}(v_s) = 1 - p_{d,0} - p_{d,1} - \alpha \cdot p_{d,2},$$

where $p_{d,k}$ is the probability that exactly k of M lines pass through a square region of side 2^{N-d} . The random image model implies that three lines intersect with zero probability at a common point. Therefore, this event may be ignored, leading to the conclusion that a region is always split if three or more lines pass through it. If only two lines pass through the region, then the region is split only if the two lines do not intersect within the region. The intersection probability α may be inferred from Theorem GP3, which implies that

$$\alpha = \frac{2\pi A}{L^2} = \frac{2\pi(2^{N-d})^2}{(4 \cdot 2^{N-d})^2} = \frac{\pi}{8}.$$

Hence,

$$(20) \quad \begin{aligned} \text{Prob}(v_s) = & 1 - \binom{M}{0} \left[1 - \left(\frac{1}{2}\right)^d \right]^M \\ & - \binom{M}{1} \left(\frac{1}{2}\right)^d \left[1 - \left(\frac{1}{2}\right)^d \right]^{M-1} \\ & - \frac{\pi}{8} \binom{M}{2} \left(\frac{1}{2}\right)^{2d} \left[1 - \left(\frac{1}{2}\right)^d \right]^{M-2}. \end{aligned}$$

In contrast to the other quadtree variants considered in this paper, the probability $\text{Prob}(v_e|v_s)$ is not 1 here. In particular, for a PM quadtree, it is possible that the splitting condition is satisfied for some node (region) at depth d but not for its parent node. For the random image model, this arises only in one case, which is when exactly two lines pass through both the node v and its parent node, with their intersection point lying inside the region corresponding to the parent node and outside the region corresponding to the node v . In this case, we have the anomalous situation that, although the node v does not exist, its corresponding region would be split (if it did exist).

Note that $P_d \leq \text{Prob}(v_s)$. We start our analysis by treating this bound, denoted by P'_d , as the splitting probability itself. The difference between the following analysis and the one carried out for the MX quadtree is that here the sum itself is an upper bound as well. Later, we also derive a tighter bound, based on an asymptotic approximation.

To obtain a bounded expression in a simpler way, let us once again, as in the MX quadtree, formulate our analysis of $E[S]$ in terms of an additional parameter β (as well as M and N), which is defined in (7). Here we shall not try to find a lower bound on the sum, as the sum itself is an upper bound.

The result of inserting (20) into (3) (i.e., a bound on $E[S]$) is decomposed into two sums \sum_1 and \sum_2 corresponding to the number of nodes at depth less than or equal to d_0 , and all the nodes at a depth greater than d_0 , respectively. In essence, we assume that the part of the tree at depth less than or equal to d_0 is complete. The value of d_0 is adjusted later so that the bounds on $E[S]$ are as tight as possible under the constraints that d_0 is an integer bounded by N and that $\beta < 1$.

Thus, we have

$$(21) \quad E[S] \leq 1 + \sum_1^N 4^d P'_{d-1} = 1 + \sum_{d=1}^{d_0} 4^d P'_{d-1} + \sum_{d=d_0+1}^N 4^d P'_{d-1} = \sum_1 + \sum_2.$$

A bound on \sum_1 is obtained as in the case of the MX quadtree:

$$(22) \quad \sum_1 = 1 + \sum_{d=1}^{d_0} P'_{d-1} 4^d \leq \sum_{d=0}^{d_0} 4^d \approx \frac{4}{3} 4^{d_0} = \frac{4}{3} \frac{M^2}{\beta^2}.$$

\sum_2 is evaluated by taking its binomial expansion to get a sum of the powers of $(\frac{1}{2})^d$ (i.e., $(\frac{1}{2})^0, (\frac{1}{2})^d, (\frac{1}{2})^{2d}, \dots$). After some algebraic manipulation, the $(\frac{1}{2})^0$ and $(\frac{1}{2})^d$ terms cancel out, and we get

$$(23) \quad \sum_2 = \sum_{d=d_0+1}^N \sum_{k=2}^M C_k \left(\frac{1}{2}\right)^{kd-k} \cdot 4^d,$$

where

$$C_k = (-1)^{k-1} \left[\binom{M}{k} - \binom{M}{1} \cdot \binom{M-1}{k-1} + \binom{M}{2} \cdot \binom{M-2}{k-2} \frac{\pi}{8} \right].$$

Changing the order of summation and separating the $k = 2$ (\sum_3) and $k > 2$ (\sum_4) cases yield

$$(24) \quad \sum_2 = \sum_3 + \sum_4,$$

$$(25) \quad \begin{aligned} \sum_3 &= \sum_{d=d_0+1}^N C_2 \left(\frac{1}{2}\right)^{2d-2} 4^d \\ &= 2 \left(1 - \frac{\pi}{8}\right) M(M-1)(N-d_0) \approx 1.215M(M-1)(N-d_0), \end{aligned}$$

$$(26) \quad \begin{aligned} \sum_4 &= \sum_{k=3}^M C_k 2^k \sum_{d=d_0+1}^N \left(\frac{1}{2}\right)^{d(k-2)} \approx \sum_{k=3}^M C_k 2^k \frac{(\frac{1}{2})^{(k-2)(d_0+1)}}{1 - (\frac{1}{2})^{k-2}} \\ &= 4 \sum_{k=3}^M \frac{C_k}{1 - (\frac{1}{2})^{k-2}} \left[\frac{\beta}{M}\right]^{k-2}. \end{aligned}$$

Now, let us examine the coefficients C_k :

$$(27) \quad \begin{aligned} C_k &= (-1)^{k-1} \left[\binom{M}{k} - \binom{M}{1} \cdot \binom{M-1}{k-1} + \binom{M}{2} \cdot \binom{M-2}{k-2} \frac{\pi}{8} \right] \\ &= (-1)^{k-1} \left[\binom{M}{k} - M \cdot \frac{k}{M} \cdot \binom{M}{k} + \binom{M}{2} \frac{k(k-1)}{M(M-1)} \cdot \binom{M}{k} \frac{\pi}{8} \right] \\ &= (-1)^{k-1} \binom{M}{k} \left[1 - k + \frac{k(k-1)\pi}{2 \cdot 8} \right]. \end{aligned}$$

By checking a few values of k , it can be shown that, for $k \geq 3$,

$$(28) \quad -0.137M^k \leq C_k \leq 0.027M^k.$$

Inserting (28) into (27) and accounting for the worst cases lead to

$$(29) \quad -1.1M^2 \frac{\beta}{1-\beta} \leq \sum_4 \leq 0.22M^2 \frac{\beta}{1-\beta}.$$

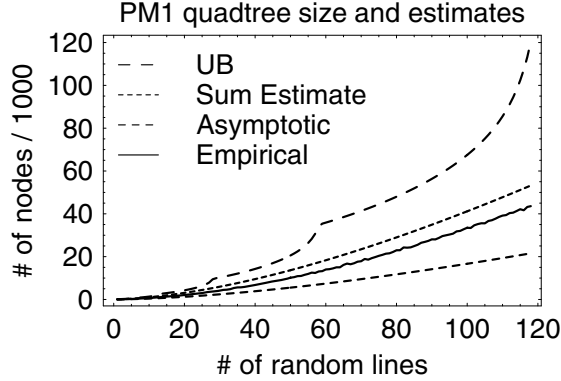


FIG. 6. The upper bound (UB), the sum estimate (Sum Estimate), which is also an upper bound, the empirical estimate (Empirical), and the asymptotic approximation (Asymptotic) on the number of nodes in a PM quadtree. The x and y axes correspond to the number of lines and nodes, respectively, for a tree of depth $N = 10$. Note the “rounded staircase-like” behavior of the upper bound resulting from our method of analysis, which calculates several bounds (each for different integer values of the d_0 parameter) retaining the tightest one.

Therefore, the contribution of \sum_4 to $E[S]$ is $O(M^2)$. Collecting the contributions of \sum_1 , \sum_3 , and \sum_4 , we have

$$(30) \quad E[S] = \sum_1 + \sum_3 + \sum_4 \leq 1.215M(M-1)N + M^2 \left[\frac{4}{3} \frac{1}{\beta^2} + 0.22 \frac{\beta}{1-\beta} - 1.215 \log_2 \frac{M}{\beta} \right].$$

Once again, d_0 is chosen to minimize (30), subject to the conditions that d_0 is an integer less than N and that β (as defined in (7)) is less than 1. Figure 6 shows the value of the upper bounds given by (30) (the uppermost curve) as a function of M at a maximal depth of $N = 10$, as well as the value of the sum (21) from which it is derived (second curve from the top). Recall that, here, the sum is also an upper bound itself. In addition, we also recall that the upper bounds in the figure are minimal in the sense that, for each value of M , upper bounds were calculated for every possible value of d_0 (subject to the constraint $\beta < 1$) and the minimal (tightest) upper bound was taken.

4.2.2. An asymptotic approximation. The bound developed above is valid but may not be tight. We now propose an asymptotic approximation which is expected to be more accurate when N is large. Denoting the parent node of v (as well as the corresponding region) by fv , we have

$$(31) \quad \begin{aligned} \text{Prob}(v_e|v_s) &= \text{Prob}((fv_e \text{ and } fv_s)|v_s) = \text{Prob}(fv_s|v_s)\text{Prob}(fv_e|(v_s \text{ and } fv_s)) \\ &= \gamma_d \text{Prob}(fv_e|(v_s \text{ and } fv_s)) \leq \gamma_d, \end{aligned}$$

where γ_d is the conditional probability $\text{Prob}(fv_s|v_s)$. As discussed above, the only possibility of this conditional event not happening is when exactly two lines cross both the region v and its parent region fv , and when these lines are arranged so that they do not intersect within v but do intersect within fv .

This probability does not have a simple expression, though, as it depends on the number of lines intersecting the region, which is distributed differently for different

depths. Note that, as the depth is increased, the probability that more than two lines intersect the region (i.e., $\sum_{k=3}^M p_{d,k}$ (see (4)) gets smaller and eventually becomes negligible. Quantitatively, the ratio of the probability that three or more lines intersect the region and the probability that exactly two lines intersect the region (i.e., $(\sum_{k=3}^M p_{d,k})/p_{d,2}$) tends to zero as d increases. As a result, the probability γ_d correspondingly decreases and reaches a constant asymptotic value $\gamma_\infty \approx 0.405$, which corresponds to the assumption that exactly two lines intersect the parent region f_v . (The value of γ_∞ was estimated by probabilistic simulation.)

Thus, with the asymptotic approximation, and relying on (31), we get a tighter bound on P_d :

$$(32) \quad P_d = \text{Prob}(v_s)\text{Prob}(v_e|v_s) \leq \text{Prob}(v_s)\gamma_\infty = P_d^\infty.$$

Using the bound P_d^∞ as P'_d in the sum (21) we get an asymptotic approximation, which is expected to better model the splitting process when the depth is high. Note that this approximation is not an upper bound, as the asymptotic approximation of the splitting probability holds only for large depths. Naturally, the asymptotic approximation is good when the majority of the nodes satisfies the above assumption but is expected to fail otherwise (i.e., when the maximal depth of the tree is small and the number of lines is large). In our experiments, we show that this is indeed the case (see Table 1). Note that technically the approximation is equal to the upper bound we obtained above (when we used $P'_d = \text{Prob}(v_s)$ in the sum) multiplied by a factor of γ_∞ . Figure 6 shows the value of the resulting asymptotic approximation as a function of M at a maximal depth of $N = 10$ (lowest curve).

4.2.3. Interpretation. The number of possible line pairs in the image is $\binom{M}{2}$. Multiplying this number by α yields the expected number of intersections. Approximating $\binom{M}{2}$ by $M^2/2$, we have that the expected number of vertices (line intersections) in the whole image is approximately $\frac{\pi}{16}M^2$. Considering only the dominant first term in (30), which is roughly proportional to $N \cdot M^2$, we see that the results of the analysis may be interpreted as confirming that both the number of vertices and the maximum depth of the tree impact the number of nodes necessary. The dependence of $E[S]$ on the number of vertices (i.e., the factor M^2) is intuitively clear, as each vertex will be stored in a separate node in the tree.

The dependence of $E[S]$ on the maximum depth of the tree (i.e., N) is less obvious. On the one hand, it could be said that, since our result is only an upper bound, it may be that the actual PM quadtree node count does not increase with the depth. However, it actually confirms a known result that, when the vertices of the line segments are constrained to lie on the grid points of a $2^n \times 2^n$ grid, the PM quadtree can be as deep as $4n$ [47]. In fact, for an image generated by the random line model, there is no constraint on the positions of the vertices (i.e., the intersection points), and thus the maximum decomposition depth can be even higher than $4 \cdot n$, as this depth depends on the locations of the vertices. Thus we see that this dependence is in agreement with the fact that, in the worst case, some of the vertices created by a random configuration of lines could appear in nodes at the maximum level. The linear dependence predicts that the probability of the occurrence of such “bad” line sets is not zero. This behavior was confirmed by our experiments, which found that the expected number of nodes in the PM quadtree increases linearly, but very slowly, with depth (see subsection 4.6). A related result is that, in most cases, the randomly generated PM quadtree is small, but in some rare cases it can be very large.

4.3. Bucket PMR_q quadtree. A bucket PMR_q quadtree represents a collection of line segments in the plane. It partitions the plane into square blocks using the splitting rule that stipulates that a block is split if both the depth of its corresponding node is less than N and more than q line segments pass through the block. Thus if the block contains q or less line segments, then it is not subdivided further and its corresponding node is a leaf. Otherwise, it is subdivided and its corresponding node has four children (see Figure 2(d)).

Note that if the expected degree of a vertex v is k , then choosing $q < k$ results in splitting the node containing v to the maximal depth. Observe that for the random image model, the degree of every vertex is 4. Thus, using a Bucket PMR_q quadtree, with $q < 4$ is not recommended. Nevertheless, for the sake of completeness, we briefly comment on these special cases below.

The Bucket PMR_0 quadtree is an MX quadtree. For the Bucket PMR_1 quadtree,

$$(33) \quad P_d < \text{Prob}(v_s) = 1 - p_{d,0} - p_{d,1}.$$

Observe that $\text{Prob}(v_e|v_s) = 1$ for all Bucket PMR quadtrees. For the Bucket PMR_2 quadtree,

$$(34) \quad P_d < \text{Prob}(v_s) = 1 - p_{d,0} - p_{d,1} - (1 - \alpha) \cdot p_{d,2}.$$

For the Bucket PMR_3 quadtree, the splitting probability is the same as in (34) with the subtraction of a term corresponding to the small probability that three line segments intersect the region, but not each other. Thus, the splitting probabilities for $q = 3$ satisfy

$$(35) \quad P_d < \text{Prob}(v_s) < 1 - p_{d,0} - p_{d,1} - (1 - \alpha) \cdot p_{d,2}$$

and are assumed to be very close to its bound.

For a Bucket PMR_2 quadtree, it is clear that the splitting probability is identical to that of the PM quadtree, apart from changing a multiplicative constant from α to $1 - \alpha$. Therefore, the expected number of nodes is given by an expression similar to (30). The same considerations hold for the Bucket PMR_3 quadtree.

While it is less obvious, the Bucket PMR_1 quadtree behaves very similarly to the PM quadtree as well. This is apparent by observing that the term $\sum 3$ (25), which dominates the number of nodes in the PM quadtree, grows by a factor of $1/(1 - \pi/8)$ if the split probability (33) is used.

The situation, however, changes dramatically when considering bucket PMR_q quadtrees with values of q equal to 4 and higher. For $q = 4$, the splitting probability satisfies

$$(36) \quad P_d < \text{Prob}(v_s) = 1 - p_{d,0} - p_{d,1} - p_{d,2}.$$

The probability P_d is smaller than the right side of the above inequality because the probabilities of some additional events that imply the absence of splitting are not included. For example, if three or four lines intersect the region, but not each other, then the region is not split. The inclusion of these contributions is complicated and is not needed for computing an upper bound. Using the upper bound $P'_d = \text{Prob}(v_s)$ in (36) as the probability P_d , and the same techniques as in subsection 4.2, we define β and d_0 as in (7) and decompose the sum corresponding to $E[S]$ into two sums, \sum_1

from (22) and \sum_2 from (23). Here, however,

$$\begin{aligned}
(37) \quad C_k &= (-1)^{k-1} \left[\binom{M}{k} - \binom{M}{1} \cdot \binom{M-1}{k-1} + \binom{M}{2} \cdot \binom{M-2}{k-2} \right] \\
&= (-1)^{k-1} \left[\binom{M}{k} - \binom{M}{1} \cdot \frac{k}{M} \cdot \binom{M}{k} + \binom{M}{2} \cdot \frac{k(k-1)}{M(M-1)} \cdot \binom{M}{k} \right] \\
&= (-1)^{k-1} \binom{M}{k} \left[\frac{(k-1)(k-2)}{2} \right],
\end{aligned}$$

implying that $C_2 = 0$ and that

$$(38) \quad -\frac{1}{8}M^k \leq C_k \leq \frac{1}{6}M^k.$$

The bounds (38) are derived by evaluating C_k for several values of k and by observing that $|C_k|$ decreases with k . The fact that $C_2 = 0$ is important, as it means that \sum_3 is 0, and thus once we bound the finite geometric series in $\frac{1}{2}$ by the infinite geometric series, the expected number of nodes will no longer depend on N . Now,

$$(39) \quad \sum_4 = \sum_{k=3}^M C_k 2^k \sum_{d=d_0+1}^N \left(\frac{1}{2}\right)^{d(k-2)} \approx \sum_{k=3}^M C_k 2^k \frac{\left(\frac{1}{2}\right)^{(k-2)(d_0+1)}}{1 - \left(\frac{1}{2}\right)^{k-2}}$$

$$(40) \quad = 4 \sum_{k=3}^M \frac{C_k}{1 - \left(\frac{1}{2}\right)^{k-2}} \left[\frac{\beta}{M}\right]^{k-2} \leq \frac{4}{3}M^2 \frac{\beta}{1 - \beta}.$$

This derivation is based on bounding the finite geometric progression with the corresponding infinite progression, expressing d_0 in terms of β and M , and performing some additional arithmetic manipulations. Therefore,

$$(41) \quad E[S] \leq \frac{4}{3}M^2 \left[\frac{1}{\beta^2} + \frac{\beta}{1 - \beta} \right].$$

Figure 7 shows the value of the upper bound given by (41) (uppermost curve) as a function of M at a maximal depth of $N = 10$ as well as the value given by the corresponding sum (middle curve). Again, we recall that the upper bounds in the figure are minimal in the sense that, for each value of M , upper bounds were calculated for every possible value of d_0 (subject to the constraint $\beta < 1$) and the minimal (tightest) upper bound was chosen. To get a clearer interpretation of the bound, select a specific value for β , say, 0.75 (which corresponds to sets of 6, 12, 24, ... lines and to d_0 values of 3, 4, 5, ..., respectively). Then, we get

$$(42) \quad E[S] \leq 6.37M^2 \quad (q = 4).$$

The bound (42) means that for a bucket PMR_q quadtrees (with $q = 4$) the expected number of nodes is proportional to the expected number of intersection points (approximately $\frac{\pi M^2}{16}$ as derived in subsection 4.2.2), henceforth referred to as *vertices*, and does not depend on the maximal depth N . Therefore, if the maximal depth is large enough, the subdivision stops before reaching the maximal depth almost everywhere. Alternatively, the $O(M^2)$ intersection points result in $O(M^2)$ line segments. Thus, an equally powerful characterization of this result is that the number of nodes is proportional to the number of line segments and does not depend on the maximal

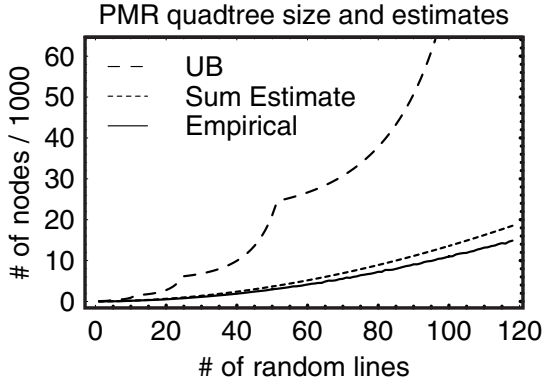


FIG. 7. The upper bound (UB), the sum estimate (Sum Estimate), and the empirical estimate (Empirical) on the number of nodes necessary to store a PMR_4 quadtree. The x and y axes correspond to the number of lines and nodes, respectively, for a tree of depth $N = 10$. Note the “rounded staircase-like” behavior of the upper bound resulting from our method of analysis, which calculates several bounds (each for different integer values of the d_0 parameter) retaining the tightest one.

depth of the tree. Higher values of q require a more complicated analysis, as the number of line segments created by the intersection of more than two lines is a random variable of more complicated statistics. In particular, we have more possibilities to consider than just the two events corresponding to the intersection or nonintersection of two lines. It is clear, however, that the splitting probability decreases as q increases, and therefore, for $q \geq 4$ the bound (41) still holds. Note that it is not possible to reduce this bound by much (even for higher values of q) since the bound on the first term, \sum_1 , remains $\frac{4}{3} \frac{M^2}{\beta^2} < \frac{4}{3} M^2$.

Regions that contain a vertex (resulting from the intersection of lines) are split if the number of line segments that are incident at this vertex is higher than q . This explains the significant change in the quadtree size when $q \geq 4$. For our random image model, all vertices have four line segments incident at them, and therefore the regions in a Bucket PMR_2 quadtree must split until the maximal depth is achieved. On the other hand, in the Bucket PMR_4 quadtree (and for values of $q \geq 4$), regions that contain a single vertex are not split, which leads to a tree whose size is independent of the maximal depth N .

4.4. PMR_q quadtree. A PMR_q quadtree represents a collection of line segments in the plane. It depends on a parameter q and is created as a dynamic result of a sequence of insertion of line segments using the splitting rule that says a block b is split once, and only once, if b is intersected by the new line segment and if b already contains q or more line segments. Thus the block is subdivided at most once when a new line segment, which intersects it, enters the structure. Clearly, this may not be enough to ensure that the number of line segments stored in each leaf node is q or less.

Note that since the PMR_q quadtree depends on the order in which the line segments are inserted into it, we must specify some order. We assume that the M infinite lines are generated in one step, followed by the insertion of the $O(M^2)$ line segments in an arbitrary order. For the case that $q \geq 4$, we may use the upper bound (41) that we obtained on the number of nodes in a bucket PMR_q quadtree to also bound the number of nodes in a PMR_q quadtree. First, note that the node set associated

with the Bucket PMR_q quadtree is a superset of the node set associated with the PMR_q quadtree, provided that the maximal depth of the tree is high enough, since the number of times a node in the Bucket PMR_q quadtree is split upon insertion is at least as large as the number of times it is split in the PMR_q quadtree. Second, observe that the maximal depth of the PMR_q quadtree for i line segments is $i - q$, as the first split happens when the $q + 1$ st line segment is inserted. Therefore, the PMR_q quadtree representation of the $O(M^2)$ line segments created by the M infinite random lines is of a maximal finite depth $O(M^2) - q$. Thus, we can construct a bucket PMR_q quadtree with a maximal depth $N = M^2$ so that, regardless of the order in which the $O(M^2)$ line segments are inserted into the PMR_q quadtree t , the nodes in t will always be a subset of the nodes in the bucket PMR_q quadtree v for the $O(M^2)$ line segments. Recall that the bound on the number of nodes in a bucket PMR_q quadtree that we obtained was independent of the depth N of the bucket PMR_q quadtree as we let N go to infinity when we computed \sum_4 in (40). Thus the bound that we obtained in (41) is also good for bucket PMR_q quadtrees of any depth. Therefore, it is also applicable to PMR_q quadtrees subject to $q \geq 4$.

Unfortunately, we cannot get a reasonable bound using this method for $q < 4$. For $q = 2$, for example, recall that the number of nodes in the Bucket PMR_2 quadtree is similar to this number in the PM quadtree, which is given by (30). Therefore, the upper bound on the number of nodes in the Bucket PMR_2 quadtree is linear in the maximum depth. The guarantee that the Bucket PMR_2 quadtree is a superset of the PMR_2 quadtree requires setting the maximal depth of the Bucket PMR_2 quadtree higher than the maximal of the PMR_2 quadtree. The latter, however, may be as high as $O(M^2)$, where M is the number of infinite random lines, thus giving an unrealistically high upper bound. Therefore, obtaining an upper bound on the number of nodes in a PMR_2 quadtree is an issue left for future research, although, as mentioned before, the case of $q \geq 4$ is more relevant from a practical standpoint, as we do not want the common situation of a road junction (i.e., when four line segments meet at a point) to cause an arbitrarily large amount of splitting. The same considerations apply for $q = 1$ and $q = 3$.

4.5. A general discussion of the bounds. The bounds developed here lead to the following asymptotic results on the expected number of nodes as a function of the number of random lines M and the level of permitted subdivision N :

$$(43) \quad \begin{aligned} \text{MX quadtree } E[S] &= O(M \cdot 2^N), \\ \text{PM quadtree } E[S] &= O(M^2 \cdot N), \\ \text{PMR}_q \text{ quadtree } E[S] &= O(M^2) \quad (q \geq 4), \\ \text{bucket PMR}_q \text{ quadtree } E[S] &= O(M^2 \cdot N) \quad (q = 2), \\ \text{bucket PMR}_q \text{ quadtree } E[S] &= O(M^2) \quad (q \geq 4). \end{aligned}$$

These results were obtained by summing the expected number of nodes at each level of the hierarchical structures. The differences are due to the different rates at which the splitting probabilities decrease as the depth increases. For example, for the MX quadtree, the probability that a node splits decreases with the depth, but does not decrease fast enough, resulting in a tree of exponential size. On the other hand, for the PM quadtree, the splitting probability decreases at a fast enough rate to offset the exponential growth of the tree, thereby resulting in a tree whose size is proportional to its depth. The same holds for the Bucket PMR_2 quadtree. For $q \geq 4$, for both the PMR_q quadtree and the bucket PMR_q quadtree, the splitting probability

decreases at an even faster rate, thereby implying that the expected number of nodes at each level decreases exponentially with the depth and that the sum converges (i.e., is independent of the depth of the tree).

The main conclusions that can be drawn from these results are as follows. For the MX quadtree, the number of nodes is proportional to the total length of the line segments. This conclusion confirms a similar result obtained by Hunter [22] and Hunter and Steiglitz [23]. For both the PM and Bucket PMR_2 quadtrees, the number of nodes can be interpreted as being proportional to the product of the number of intersections among the lines (i.e., the original lines in the random image model or, alternatively, the vertices of the resulting line segments) and the maximal depth of the tree. For both the PMR_q quadtree and the Bucket PMR_q quadtree with node capacities $q \geq 4$, the number of nodes is proportional to the number of line segments (recall that there are $O(M^2)$ intersection points for the M lines, resulting in $O(M^2)$ line segments). It also appears that, for the PMR_q ($q \geq 4$) quadtree, almost everywhere, the subdivision stops before the maximal depth, provided, of course, that the density of the lines (i.e., the M random lines) does not make the tree almost full.

To get the actual values of the bounds (in contrast to the orders of magnitude summarized above) we use the exact upper bounds as given in (15), (30), and (41), which depend on a parameter β . It is worth re-emphasizing that the values d_0 and β are not a part of the random image model. They are just parameters used to simplify the expression of the bounds. In order to apply these bounds, it is required to choose a value of β which minimizes them while satisfying relation (7) (with d_0 being an integer bounded by N). Fixing the value of β at some constant (e.g., 0.75) gives a bound, which may not be the tightest but is still useful for understanding the behavior of the size of the data structure. From a strict theoretical standpoint, such an arbitrary choice of a value for β is not justified because it usually implies a noninteger value for d_0 . Note also that the upper bounds contain negative terms which reduce the bounds and make them tighter. These negative terms compensate for nodes which are counted twice in other (positive) terms. For example, nodes in the PM quadtree at a depth less than d_0 are accounted for both by the $\sum_1 = \frac{4}{3} \frac{M^2}{\beta^2}$ term and also by the $M^2 \cdot N$ term. The negative term $-M^2 \log_2 \frac{M}{\beta} \approx -M^2 d_0$ compensates for this situation. Note also that these bounds hold for all values of M and N . However, they become trivial when $M \geq 2^N$.

The bounds in this paper were computed under the assumption of a particular image model. We conjecture that the results apply also to more general images. In section 5 we examine several methods for inferring the size of quadtrees that represent real maps and test them experimentally. In essence, we characterize the map by some property, which may be the total length of its constituent line segments, the number of vertices, etc., and use this property to specify a class of random images which share the same property (in an expected value sense). Next, we conjecture that the number of quadtree nodes required to represent the real map is equal to the expected number of nodes required to represent a random map from that class.

4.6. Some experimental results for instances of the random model. We conducted several experiments with synthetic and real data. In this section we describe the tests that were made with synthetic data. They were aimed at determining how close the upper and lower bounds on the expected storage costs come to the actual storage costs when using random data. Section 5 describes the results of tests with real data.

In these experiments, we built the MX, PM, and Bucket PMR_4 quadtrees of

TABLE 1

Comparison of the result of expression (3) in plain form (MODEL) and asymptotic form ($MODEL_\infty$), upper bounds (UB), lower bounds (LB), and the actual (experimental) number of nodes (ER) for M random lines and a maximal depth N in an MX, PM, and Bucket PMR_4 quadtrees.

M	N	MX				PM			
		MODEL	UB	LB	ER	MODEL	UB	$MODEL_\infty$	ER
25	10	94.8K	97.5K	90.7K	94.9K	3.88K	5.35K	1.57K	2.58K
50	10	179K	189K	162K	181K	12.8K	19K	5.18K	9.43K
75	10	255K	269K	225K	256K	25.1K	43.2K	10.2K	19.3K
100	10	325K	365K	259K	325K	39.9K	64.9K	16.2K	32.2K
25	14	1.63M	1.63M	1.62M	1.62M	6.78K	8.27K	2.74K	2.94K
50	14	3.23M	3.24M	3.21M	3.24M	24.6K	30.9K	9.95K	11.7K
75	14	4.82M	4.83M	4.79M	4.80M	51.6K	70.2K	20.9K	25.5K
100	14	6.39M	6.43M	6.32M	6.37M	87K	113K	35.2K	45.2K

M	N	Bucket PMR_4		
		MODEL	UB	ER
25	10	0.846K	4.34K	0.669K
50	10	7.5K	32.5K	6.01K
75	10	3.38K	17.4K	2.73K
100	10	13.1K	69.5K	10.6K
25	14	0.863K	4.34K	0.681K
50	14	3.52K	17.4K	2.83K
75	14	7.97K	32.5K	6.35K
100	14	14.2K	69.5K	11.4K

several depths N , using random synthetic data created by the random image model described in section 3. For each case, several instances of each random image were created and the average quadtree size was calculated. The results are summarized in Table 1 (see also Figure 5). They usually agree with the analytical predictions, and, in particular, the nonasymptotic bounds for all of the quadtree variants always hold.

We also evaluated the expression (3) for the values of the number of lines M and the maximum depth N of the quadtrees. We found that while this sum closely approximates the actual expected number of nodes for the MX quadtree, it only bounds the number of nodes for the PM and the Bucket PMR_4 quadtrees. This is expected because, for these trees, the P'_d expressions, which we used, are bounds of the probabilities and not the probabilities themselves. The PM quadtree asymptotic approximation, which corrects every bound by the γ_∞ factor (see subsection 4.2.2), yields a more accurate estimate for trees with a high depth and a low number of lines (as expected). For the Bucket PMR_4 quadtree, the sum is a more accurate estimate (as it errs only by about 25%).

The upper bounds obtained for the MX quadtree were consistently very close to the observed node counts. In contrast, the upper bounds for the PM and Bucket PMR_4 quadtrees consistently exceed the observed node counts by factors as high as 3 and 7, respectively. This difference can be attributed to the simplifications made in the bound derivation process. As mentioned above, we did not attach much significance to obtaining tighter bounds, as they are not used for predictions but, instead, only for performing a qualitative comparison between the different quadtree representations.

We also conducted some experiments to test the performance of the PM quadtree under “asymptotic” conditions—that is, when N is relatively large and M is not too high. These experiments were undertaken to determine whether or not the PM quadtree grows linearly with the maximal depth N (like the upper bound that we found). The results were still inconclusive; when examining the number of nodes, we found that the average number (over 1000 random trials) increases slowly but steadily with maximal depth, but the high variance still does not allow us to conduct a decisive statistical test. We also examined histograms of the actual maximal depth

TABLE 2

Description of the TIGER files maps (see Figure 8) used in the experiments as well as the corresponding actual storage requirements for the MX, PM, and Bucket PMR₄ quadtrees.

Map Name	Depth	Vertices	NSV	NormL	Segments	Number of nodes		
						MX	PM	PMR ₄
Falls Church	10	448	317	16.77	638	76869	4105	1317
Falls Church	12	448	317	16.77	638	336873	4477	1349
Falls Church	14	448	317	16.77	638	1387557	4633	1381
Falls Church	16	448	317	16.77	638	5600821	4681	1413
Alexandria	10	4074	2123	49.89	5380	191469	25249	9709
Alexandria	12	4074	2123	49.89	5380	915025	28929	10105
Alexandria	14	4074	2123	49.89	5380	3873949	30413	10429
Alexandria	16	4074	2123	49.89	5380	15774517	31061	10753
Arlington	10	6657	3978	67.91	9205	242373	43913	17637
Arlington	12	6657	3978	67.91	9205	1234449	54753	18677
Arlington	14	6657	3978	67.91	9205	5339437	58949	19481
Arlington	16	6657	3978	67.91	9205	21891973	61277	20281
Howard	10	15009	5283	57.58	17419	191861	63361	29321
Howard	12	15009	5283	57.58	17419	1031317	95945	32921
Howard	14	15009	5283	57.58	17419	4563069	111169	33937
Howard	16	15009	5283	57.58	17419	18866341	118305	34713
DC	10	12818	8805	107.99	19183	332589	73477	35377
DC	12	12818	8805	107.99	19183	1805965	92153	37813
DC	14	12818	8805	107.99	19183	7971497	99093	39685
DC	16	12818	8805	107.99	19183	32905753	103297	41533
Calvert	10	29174	4690	60.39	31143	213965	88769	44057
Calvert	12	29174	4690	60.39	31143	1094453	125053	48129
Calvert	14	29174	4690	60.39	31143	4734589	133141	48493
Calvert	16	29174	4690	60.39	31143	19402105	135241	48665
Prince George's	10	50161	18055	117.55	59551	315289	157977	88485
Prince George's	12	50161	18055	117.55	59551	1937553	277209	104993
Prince George's	14	50161	18055	117.55	59551	8993017	318889	107889
Prince George's	16	50161	18055	117.55	59551	37751493	334265	109825
Montgomery	10	79822	19793	118.74	90022	299601	186265	115693
Montgomery	12	79822	19793	118.74	90022	1927197	365701	145389
Montgomery	14	79822	19793	118.74	90022	9068057	424869	149613
Montgomery	16	79822	19793	118.74	90022	38212101	449905	151913

as a function of the allowed maximal depth. Here we found that even for depths as large as 30 we still had trials (and corresponding random collections of 25 lines), where the PM quadtree had nodes at this maximal depth. In contrast, this never happened for the PMR quadtree.

5. Predicting storage requirements for real data. In this section we show that the expected storage predictions, derived for the random image model, are also useful when real data is considered.

We conducted our tests using real data corresponding to U.S. city and county road maps that are part of the TIGER files used by the U.S. Census Bureau (see Figure 8). We used maps ranging from a small map having only 585 road segments (Falls Church, Virginia) to the largest map having 39,719 segments (Montgomery County, Maryland). The actual data for the maps, such as the depth (N), number of vertices, segments, nonshape vertices (abbreviated *NSV* and described below), the normalized length (abbreviated *NormL* and equal to the total length of the line segments divided by 2^N), and the number of nodes in the MX, PM, and Bucket PMR₄ quadtrees, are given in Table 2.

Our approach to applying the expected node count predictions to a real map r depends on finding, for each map r , a class c of a random line image which shares some property with r . The expected number of nodes required to represent a random image from class c is taken to be the estimate for the number of nodes required to represent the map r .

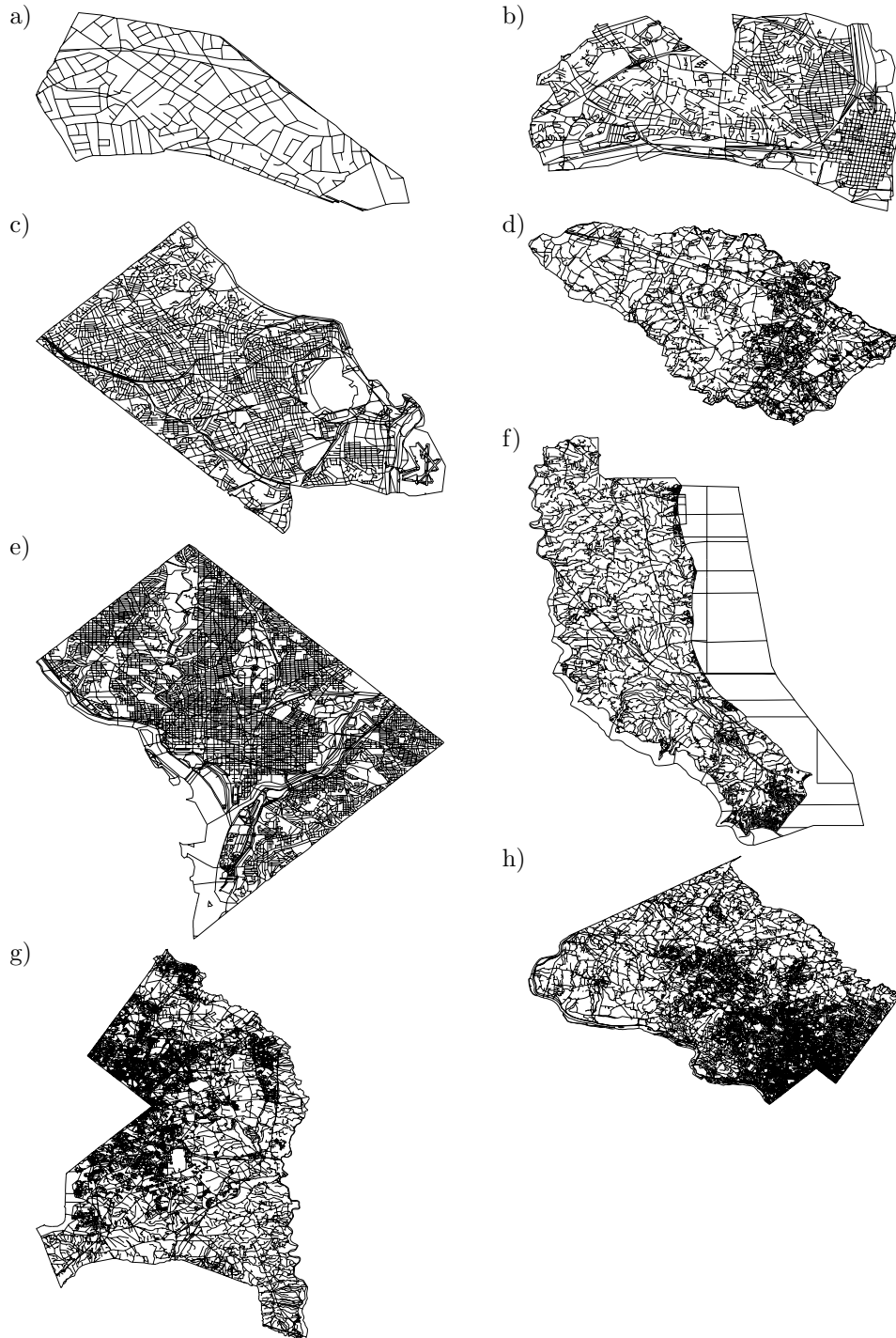


FIG. 8. Eight maps used to test the estimates on the number of nodes in the representing quadtrees.

In most cases, the equivalent random image to a given map r is specified, as in section 3, by the effective number of lines \hat{M} , which is inferred, in a number of alternative ways (termed *estimators* and described below), from r . Another degree of freedom in the specification of the equivalent random image is gained if we account for sparse or empty image parts. Therefore, the equivalent random image is specified in two stages. First, a random image is specified by \hat{M} . Second, only a fraction of this image is retained, while the rest is considered to be empty. A normalization factor is inferred from the image r and specifies the fraction that is retained. Thus, we assume that the number of nodes is directly proportional to the portion of the area that is nonempty, and we use this normalization factor to obtain the number of nodes in the equivalent class. For most of the estimators, no normalization is done and the images of the equivalent class are just random images resulting from the random image model defined in section 3. The parameters specifying the class of random images are estimated from other parameter values of the real map r such as the total length L_{map} , the total number of vertices V_{map} , and the total number of segments S_{map} .

The first estimator that we consider is termed an *L-based estimator*. This estimator is based on measuring the total length L_{map} of the line segments. Recall that, for the random images created by the random image model, the expected total line length $E[L]$ is $\pi/4 \cdot M \cdot 2^N$. This is the result of (16), which follows directly from Theorem GP2. Therefore, the number of lines in every one of the random images which share the “expected line length” property with the given map is estimated by $\hat{M} = (L_{map}/2^N) \cdot (4/\pi)$. For this estimator, there is no area normalization (that is, the equivalent random image is assumed not to contain empty regions). For example, the total length of the line segments in the smallest map (i.e., Falls Church, Virginia shown in Figure 8(a)) that we used is $16.77 \cdot 2^N$, which yields $\hat{M} \approx 21$ (the length is normalized relative to the side of the map). Note that this has the effect of converting the line segments of the test image to a different number of infinite lines, the intersection of which, with the space in which they are embedded, has the same total length (in an expected value sense).

The second estimator that we consider is termed a *V-based estimator*. It uses the number of vertices V_{map} in the map to estimate the effective number of lines \hat{M} by treating all vertices as intersection points between random lines. Recalling that the expected number of intersection points between the lines of the random image model is $E[V] = \pi M^2/16$, we have $\hat{M} = \sqrt{16V_{map}/\pi}$. Here, again as in the case of the L-based estimator, no area normalization is performed. Note, however, that most vertices (termed *shape points* [7]) in a typical map image are the results of a piecewise polygonal approximation of a curve, thereby implying that only two line segments meet at them. This is in contrast to the intersection points in a random image, which are true intersections (i.e., they correspond to the intersections of pairs of random lines). A simple heuristic, which we use here, deletes these degree-2 vertices from the total vertex count and yields the *NSV estimator*. Clearly, there are many cases for which this heuristic is nonapplicable. For example, we could not apply it to a non-self-intersecting curve (e.g., a spiral), as vertex deletion would predict that the representation requires just a single node (which is clearly erroneous).

The third estimator that we consider is termed an *S-based estimator*. Like the L-based and V-based estimators, it is based on replacing the given map with a random line image of the same size, and no area normalization is performed. Again, we calculate an effective number of random lines \hat{M} , but this time it is based on the

number of line segments. We assume that each vertex corresponds to the intersection of two random lines, and hence results in four line segments (also termed *edges* or *segments*). Therefore, each vertex has degree 4. Each line segment is incident at two vertices. This is approximately true, as only a small fraction of the line segments (i.e., $2M$) are incident at just one vertex, as the other endpoint of the line is on the boundary of the image. Therefore, set the number of incidences (i.e., the sum of the degrees of the vertices), which is four times the expected number of vertices (i.e., $4 \cdot \pi M^2/16$), to two times the number of edges (i.e., $2 \cdot S_{map}$) and solve for \hat{M} , which is equal to $\sqrt{8S_{map}/\pi}$.

The fourth estimator that we consider is termed a *d-based estimator*. This estimator is based on replacing the given map with a (usually smaller) random line image having the same number of vertices and average line segment length (termed *density* here). The expected segment length in the random line image is crudely approximated as the ratio between the expected length of the part of a random line included in the image and the expected number of vertices on the line. Using Theorem GP2 from geometric probability, we know that the expected length of the part of a random line included in the image is $\pi \cdot 2^N/4$, while the expected number of vertices in the map is $\pi M^2/16$. Since each vertex corresponds to the intersection of two lines and hence lies on two lines, the expected number of vertices per line is $(\pi M^2/16)/(2M) = \pi M/8$. Therefore, the expected segment length is $(\pi \cdot 2^N/4)/(\pi M/8) = 2 \cdot 2^N/M$. Equating this expected segment length to the average segment length of the given map, calculated simply as the ratio between the total length L_{map} and the number of segments S_{map} (i.e., L_{map}/S_{map}), and solving for M yield an estimate \hat{M} on the effective number of lines, which is equal to $2 \cdot S_{map} \cdot 2^N/L_{map}$.

Unlike instances of the random image model, real maps tend to be highly nonuniform, and, in particular, to have a large proportion of short segments and large empty “white” regions. This implies that the value of the effective number of lines \hat{M} calculated for the d-based estimator above leads to node number estimates which are much higher than the actual ones. Therefore, we have chosen to compensate for this deviation by imposing the additional natural constraint that the total number of NSVs in the actual map is equal to the expected number of vertices in the random line images. This constraint, which was also used to obtain the effective number of lines for the V-based estimator, is now used as an area normalization factor to specify the nonuniform class of random images. In particular, every one of these images is equal to the random lines image in one region and is empty in the rest of it. The area of the “busy” part is specified to be $\frac{NSV_{map}}{(\pi \cdot M^2/16)}$ and is usually smaller than 1. Note that the expected density remains the same. Since the random image model defined in section 3 is uniform, we can assume that the expected number of nodes representing every region is proportional to its area, and thus the node count is reduced by the aforementioned factor.

In order to estimate the number of nodes in the quadtrees representing the actual maps, we round the different values of the estimate \hat{M} and insert this estimate into the basic sum (3) for the expected number of nodes together with the appropriate node splitting probability (which depends only on the quadtree type). These results are tabulated in Table 3. Notice that we do not tabulate the S-based estimator, as it is very similar to the V-based estimator, and the data bears this out. In particular, the S-based estimator relies on the number of line segments. This number is related to the number of vertices, which is the basis of the V-based estimator. Upper bounds, which have a more compact form and do not require the evaluation of a sum, may be

TABLE 3

Predicted storage requirements for the MX, PM, and bucket PMR₄ quadtrees using the three estimators. The numbers given in the table are the ratios between the predicted requirements using the estimator and the actual ones given in Table 2.

Map Name	Depth	L-estimator			V-estimator			d-estimator		
		MX	PM	PMR ₄	MX	PM	PMR ₄	MX	PM	PMR ₄
Falls Church	10	1.05	0.50	0.45	1.90	1.57	1.65	0.94	1.31	1.63
Falls Church	12	1.00	0.61	0.45	1.88	1.98	1.65	0.97	1.75	1.67
Falls Church	14	0.99	0.73	0.44	1.87	2.44	1.63	0.98	2.23	1.65
Falls Church	16	0.98	0.86	0.43	1.87	2.94	1.59	0.99	2.74	1.62
Alexandria	10	1.16	0.57	0.57	1.75	1.28	1.45	0.71	0.95	1.32
Alexandria	12	1.09	0.72	0.57	1.72	1.70	1.49	0.78	1.40	1.47
Alexandria	14	1.06	0.89	0.56	1.71	2.17	1.47	0.81	1.89	1.48
Alexandria	16	1.06	1.07	0.54	1.71	2.66	1.44	0.82	2.39	1.44
Arlington	10	1.18	0.54	0.55	1.77	1.22	1.44	0.78	0.91	1.30
Arlington	12	1.07	0.64	0.55	1.70	1.55	1.50	0.83	1.29	1.47
Arlington	14	1.03	0.79	0.54	1.69	1.97	1.47	0.87	1.73	1.47
Arlington	16	1.02	0.95	0.52	1.68	2.40	1.42	0.88	2.18	1.43
Howard	10	1.30	0.28	0.24	2.50	1.06	1.14	0.40	0.50	0.78
Howard	12	1.09	0.27	0.23	2.32	1.13	1.13	0.51	0.73	1.03
Howard	14	1.03	0.31	0.22	2.27	1.35	1.13	0.57	1.00	1.10
Howard	16	1.01	0.36	0.22	2.25	1.63	1.11	0.60	1.30	1.11
DC	10	1.26	0.69	0.67	1.74	1.37	1.51	0.86	1.03	1.33
DC	12	1.12	0.86	0.69	1.67	1.83	1.63	0.93	1.55	1.57
DC	14	1.09	1.10	0.67	1.66	2.41	1.61	0.97	2.14	1.59
DC	16	1.08	1.33	0.65	1.66	2.99	1.55	0.98	2.73	1.54
Calvert	10	1.22	0.22	0.18	2.15	0.70	0.68	0.13	0.19	0.32
Calvert	12	1.08	0.23	0.17	2.08	0.79	0.69	0.22	0.39	0.57
Calvert	14	1.04	0.29	0.17	2.07	1.02	0.70	0.27	0.64	0.67
Calvert	16	1.03	0.35	0.17	2.07	1.28	0.71	0.30	0.90	0.70
Prince George's	10	1.42	0.37	0.32	2.32	1.08	1.14	0.35	0.42	0.63
Prince George's	12	1.14	0.34	0.30	2.12	1.11	1.18	0.48	0.68	1.01
Prince George's	14	1.06	0.40	0.30	2.07	1.41	1.20	0.56	1.03	1.16
Prince George's	16	1.03	0.49	0.29	2.06	1.78	1.20	0.60	1.41	1.19
Montgomery	10	1.50	0.32	0.25	2.51	0.98	0.94	0.20	0.24	0.35
Montgomery	12	1.15	0.26	0.22	2.22	0.91	0.93	0.30	0.45	0.71
Montgomery	14	1.05	0.30	0.22	2.15	1.15	0.95	0.38	0.74	0.89
Montgomery	16	1.02	0.37	0.21	2.13	1.43	0.95	0.42	1.05	0.94

obtained by inserting the rounded estimate \hat{M} directly into the upper bounds (15), (30), and (41), although we do not tabulate them here.

The particular estimators described have a number of inherent limitations. For example, suppose that the scale of the line segments (roads) of the map is lowered by a factor of 2 so that all the roads are totally embedded in the NW quadrant of the original map image, while the rest of the scaled map image is empty. In this case, if the depth is high enough, it is likely that the number of leaf nodes in PM and Bucket PMR₄ quadtrees will stay the same, while that in the MX quadtree will decrease significantly. However, the total length of the lines in the quadtree of the scaled-down map will be off by a factor of 2, thereby implying that the L-based estimator is likely to be inaccurate for the PM and bucket PMR₄ quadtrees. This is most relevant for maps containing many line segments in a small area and appears to dampen the suitability of the L-based estimator for arbitrary images, although it does seem to work for images that span most of the space in which they are embedded.

From our experiments, the L-based estimator seems to perform best for the MX quadtrees, while the V-based estimator seems to work best for the PM and PMR₄ quadtrees. We constructed the d-based estimator to try to improve further on the estimates for the PM and PMR₄ quadtrees but found that it does not improve on the V-based estimator, and sometimes even does worse. The good performance of the L-based estimator for the MX quadtree was not surprising, as it confirms the original result of the analysis of Hunter [22] and Hunter and Steiglitz [23], who found a propor-

tionality to the perimeter of the image. The correlation between the estimated and actual values that we observed is noteworthy, considering that the number of nodes in the maps ranged between 77,000 and 38 million (i.e., a factor of 500). Nevertheless, we believe that a more detailed examination of the differences between the actual and predicted storage requirements, as well as other properties, of hierarchical spatial data structures is an extremely interesting open problem.

We were also interested in testing the validity of the asymptotic results given in (43) on the expected number of nodes as a function of the number of line segments in real images and the level of permitted subdivision. To do that, we interpret these results in terms of an actual map parameter—for example, by replacing the number of infinite lines M with its estimate, as done above. For example, using the S-based estimator indicates that the size of the PMR quadtree should be proportional to the number of map segments and independent of the maximal depth (as long as both are large). Note that this expectation is verified (or refuted) only on the basis of the real map data and not on the basis of any modeling assumptions, although we were indeed led to it by the random line model analysis.

We first examine the MX quadtree. Figure 9 shows the ratios of the node count to the length of a side of the image (i.e., 2^N) as a function of the depth (i.e., N) for the different maps, which are close to being constant (i.e., horizontal lines) as expected. Figure 10 shows the ratios of the node count to the square root of the number of line segments as a function of the number of line segments (i.e., M^2) for the different depths. These curves are close to being constant (i.e., horizontal lines) when the number of line segments is large enough. This is expected when the S-estimator (which stipulates that the number of nodes is proportional to the square root of the number of segments) is used for M and implies that the asymptotic estimate is useful for predicting the quadtree size for a wide range of maps. We used a logarithmic scale in Figure 10 to illustrate a similar relative deviation in the ratios for the different depths as the size of the data increases.

Next, we examine the PM quadtree. Figures 11 and 12 show the ratio of the node count to the NSVs and to the number of line segments (i.e., M^2), respectively, as a function of the depth for the different maps, which are close to being constant (i.e., horizontal lines). This means that the node count is independent of the depth. This is contrary to the prediction of the asymptotic analysis and to the existing worst cases that arise when the vertices of the line segments are constrained to lie on grid points [47]. This difference may be explained by observing that, for images generated using the random image model, factors that lead to the maximum depth (e.g., two vertices or nonintersecting lines being very close to each other, or a vertex and a line being very close [49]) are more likely to arise. For road networks, on the other hand, it is unlikely that a pair of intersections 10 cm from each other will be specified. In such a case, these intersections will be merged to a higher degree vertex, which is not split for the PM quadtree. Thus, it seems that there is room for a better model for road networks (and maybe for other types of real data), which takes such merging processes into account. This subject is left for future research.

Finally, we examine the PMR₄ quadtree. Figures 13 and 14 show the ratio of the node count to the NSVs and to the number of line segments (i.e., M^2), respectively, as a function of the depth for the different maps, which we expect to be constant (i.e., horizontal lines), especially for the larger maps. At lower depths, the ratios increase with depth for a particular map since the segment counts are constant and the number of nodes does increase with depth until converging once the decomposition rule can

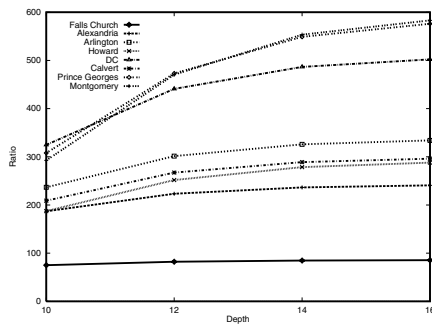


FIG. 9. Ratio of MX quadtree nodes to side length.

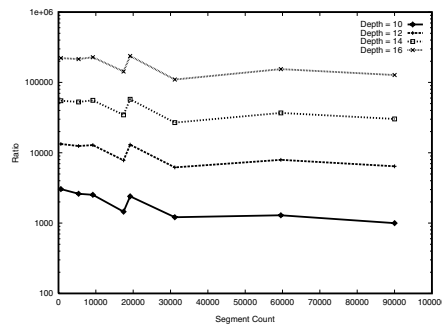


FIG. 10. Ratio of MX quadtree nodes to square root of segment count.

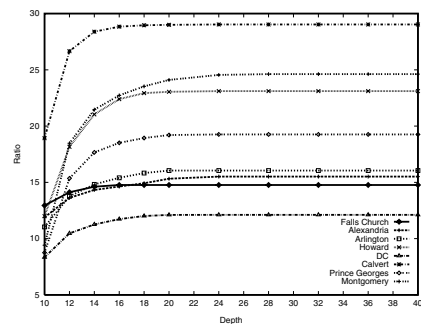


FIG. 11. Ratio of PM quadtree nodes to NSV.

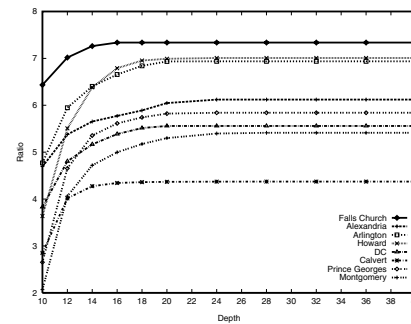


FIG. 12. Ratio of PM quadtree nodes to the segment count.

no longer be applied. It is interesting to observe that the lines in Figures 13 and 14 are not quite horizontal (i.e., representing a constant function) in the sense that they have a small positive slope. This is because the line segments in the maps are not really formed by random infinite lines. Thus, it is not true that the probability that more than two infinite lines intersect at a point is zero. In particular, we find that in our maps there are instances when more than four line segments meet at a point, and hence the number of nodes really grows linearly with depth (since the decomposition rule is still applicable, and, in fact, will always be applicable in this case), although this growth is not substantial in our graphs at higher depths. Experiments with larger values of q verified that the number of nodes does in fact converge as the depth increases. This can be seen in Figure 15 for $q = 12$. Figure 5 shows the ratio of the node count to the segment count (i.e., M^2) versus the segment count at depth 16 for $q = 4$ and $q = 12$. The ratios are all within 6% of their average value. Figure 5 also reveals a general trend in which the ratios decrease as the maps get larger. We used a logarithmic scale to illustrate a similar relative deviation in the ratios for the different values of q as the size of the maps increases.

To more clearly see the effect of the existence of points, where more than four line segments intersect, consider the map of Washington, D.C. (Figure 8). From Figure 11, we can see that the number of nodes increases by a ratio of about 7:4 when the depth is changed from 10 to 40. From Table 2, we can see that this amounts to an increase of about $(7/4 - 1) \cdot 35,000 = 26,250$ vertices. Now, let W be the number of

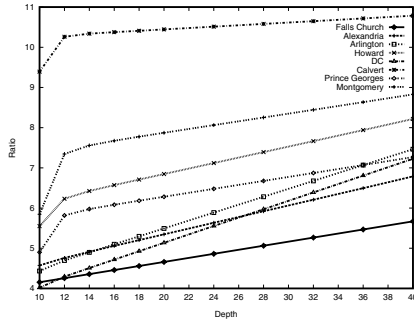


FIG. 13. Ratio of PMR quadtree nodes to NSV, $q = 4$.

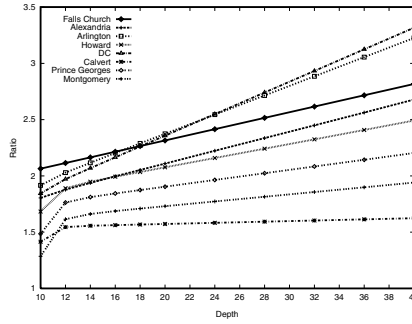


FIG. 14. Ratio of PMR quadtree nodes to segment count, $q = 4$.

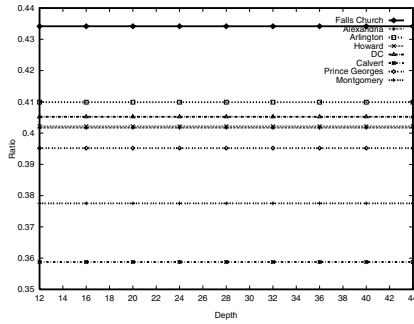


FIG. 15. Ratio of PMR quadtree nodes to segment count, $q = 12$.

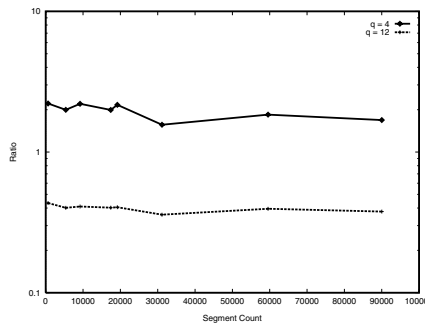


FIG. 16. Ratio of PMR quadtree nodes to segment count versus segment count at a depth of 16.

vertices associated with the intersection of more than four roads. Each such vertex leads to a split of its corresponding region until the maximal depth is reached. At every depth, four nodes are added. Therefore, when the depth is changed from 10 to 40, the number of nodes that are added is $W \cdot (40 - 10) \cdot 4 = 26,250$, implying that the number of such high degree vertices is about 220. Given that the Washington, D.C. map has 8805 NSVs (i.e., vertices of degree greater than 20), the fraction of such high degree vertices is about $220/8805 = 0.025$. While this number is larger than 0 (the value predicted by our model), it does not appear to change the predictions by much, as is apparent from the relatively modest increase in the node count as the depth increases. Note that for the other maps considered the relative increase in the node count, as well as the fraction of high degree vertices, is much lower. Observe also that the range of maximum tree depths considered in these graphs is much larger than in any reasonable application. In particular, if the original map is embedded in a 100×100 km square area, then a maximum depth of 40 amounts to a resolution of 0.1 micron. Therefore, for a more reasonable maximum depth, the variation in the predicted node count will be much lower even if, say, 0.025 of the vertices have high degree.

6. Concluding remarks. The analysis of the space requirements of a number of trie-based hierarchical geometric data structures for storing large collections of line segments was investigated using a random image model. An appropriate model was developed for each of these structures and estimates of $E[S]$, the expected number of nodes, were found for them. Future work includes the investigation of the use of these estimates in a cost model by a query optimizer to generate an appropriate query evaluation plan in a spatial database application. The analysis presented here is also of interest because it uses a detailed explicit model of the image, instead of relying on modeling the branching process represented by the tree and leaving the underlying image unspecified. The behavior of these expected values is intuitively and concisely expressed by analytic upper and lower bounds. Other directions for future research include the application of the geometric probability approach to additional data types besides line segments (e.g., points, polygons, surfaces, solids, etc.), as well as alternative trie-based spatial data structures.

We have demonstrated that these estimates, derived for a particular random image model, are applicable to real data. Specifically, in the case of line map images, we provided estimators which are based on simple characterizations of the map data, and which enabled us to successfully apply the results of the analytic model to real data and to obtain reasonably accurate and useful results. This was verified, however, only for map data, and characterizing collections of other types of line segments, or even more general types of spatial information, is still an open problem, and thus a subject for further research.

Our results can be used to justify claims on the qualitative differences between the different alternative spatial data structures. For example, we showed that the bucket (and conventional) PMR_q quadtree for $q \geq 4$ is superior to the PM quadtree in terms of the number of nodes that are required. The problem with the PM quadtree is that, although its behavior is usually acceptable, there are cases in which it requires much space due to certain point and line configurations. This follows from our analysis and simulations, as well as from confirming earlier observations on the possible worst-case behavior of the PM quadtree [47].

Perhaps our most important result is showing that the space requirements of the Bucket PMR_q and PMR_q ($q \geq 4$) quadtrees are asymptotically proportional to the number of line segments. This was shown theoretically for a random image model and was also found to hold for random data and real map data. This is quite significant as it enables us to predict the number of nodes required by this representation, and, most important, to show that it is independent of the maximum depth of the tree. It is thus not surprising that the PMR_q quadtree is useful in experimental systems (e.g., QUILT [53]) as well as commercial systems (e.g., United Parcel Service (UPS) [4]).

Acknowledgments. We thank S. K. Bhaskar and Gary D. Knott for suggestions about the evaluation of the sums.

REFERENCES

- [1] C. H. ANG, *Applications and Analysis of Hierarchical Data Structures*, Tech. report TR-2255, Department of Computer Science, University of Maryland, College Park, MD, 1989.
- [2] W. G. AREF AND H. SAMET, *Optimization strategies for spatial query processing*, in Proceedings of the 17th Annual International Conference on Very Large Data Bases (VLDB), G. Lohman, ed., Barcelona, Spain, September 1991, pp. 81–90.
- [3] N. BECKMANN, H. P. KRIEGEL, R. SCHNEIDER, AND B. SEEGER, *The R*-tree: An efficient and robust access method for points and rectangles*, in Proceedings of the ACM SIGMOD Conference, Atlantic City, NJ, June 1990, pp. 322–331.
- [4] R. BONEFAS, *Personal communication*, 1991.

- [5] R. DE LA BRIANDAIS, *File searching using variable-length keys*, in Proceedings of the IRE/IEEE/ACM Western Joint Computer Conference, San Francisco, CA, 1959, pp. 295–298.
- [6] T. BRINKHOFF, H. P. KRIEGEL, R. SCHNEIDER, AND B. SEEGER, *Multi-step processing of spatial joins*, in Proceedings of the ACM SIGMOD Conference, Minneapolis, MN, June 1994, pp. 197–208.
- [7] BUREAU OF THE CENSUS, *TIGER/Line Census Files*, 1990 Technical Documentation, Washington, DC, 1991.
- [8] C. R. DYER, *The space efficiency of quadtrees*, Comput. Graphics Image Process., 19 (1982), pp. 335–348.
- [9] C. FALOUTSOS AND I. KAMEL, *Beyond uniformity and independence: Analysis of R-trees using the concept of fractal dimension*, in Proceedings of the ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS), Minneapolis, MN, May 1994, pp. 4–13.
- [10] C. FALOUTSOS, H. V. JAGADISH, AND Y. MANOLOPOULOS, *Analysis of the n-dimensional quadtree decomposition for arbitrary hyperrectangles*, IEEE Trans. Knowledge Data Engrg., 9 (1997), pp. 373–383.
- [11] C. FALOUTSOS, T. SELIS, AND N. ROUSSOPOULOS, *Analysis of object oriented spatial access methods*, in Proceedings of the ACM SIGMOD Conference, San Francisco, CA, May 1987, pp. 426–439.
- [12] R. A. FINKEL AND J. L. BENTLEY, *Quad trees: A data structure for retrieval on composite keys*, Acta Inform., 4 (1974), pp. 1–9.
- [13] P. FLAJOLET, G. GONNET, C. PUECH, AND J. M. ROBSON, *Analytic variations on quadtrees*, Algorithmica, 10 (1993), pp. 473–500.
- [14] P. FLAJOLET AND T. LAFFORGE, *Search costs in quadtrees and singularity perturbation asymptotics*, Discrete Comput. Geom., 12 (1994), pp. 151–175.
- [15] P. FLAJOLET AND C. PUECH, *Partial match retrieval of multidimensional data*, J. ACM, 33 (1986), pp. 371–407.
- [16] E. FREDKIN, *Trie memory*, Comm. ACM, 3 (1960), pp. 490–499.
- [17] G. GRAEFE, *Query evaluation techniques for large databases*, ACM Comput. Surveys, 25 (1993), pp. 73–170.
- [18] A. GUTTMAN, *R-trees: A dynamic index structure for spatial searching*, in Proceedings of the ACM SIGMOD Conference, Boston, MA, June 1984, pp. 47–57.
- [19] E. G. HOEL AND H. SAMET, *Data-parallel spatial join algorithms*, in Proceedings of the 23rd International Conference on Parallel Processing, vol. 3, St. Charles, IL, 1994, pp. 227–234.
- [20] E. G. HOEL AND H. SAMET, *Performance of data-parallel spatial operations*, in Proceedings of the 20th Annual International Conference on Very Large Data Bases (VLDB), Santiago, Chile, September 1994, pp. 156–167.
- [21] M. HOSHI AND P. FLAJOLET, *Page usage in a quadtree index*, BIT, 32 (1992), pp. 384–402.
- [22] G. M. HUNTER, *Efficient Computation and Data Structures for Graphics*, Ph.D. dissertation, Department of Electrical Engineering and Computer Science, Princeton University, Princeton, NJ, 1978.
- [23] G. M. HUNTER AND K. STEIGLITZ, *Operations on images using quad trees*, IEEE Trans. Pattern Anal. Mach. Intell., 1 (1979), pp. 145–153.
- [24] C. L. JACKINS AND S. L. TANIMOTO, *Oct-trees and their use in representing three dimensional objects*, Comput. Graphics Image Process., 14 (1980), pp. 249–270.
- [25] G. KEDEM, *The quad-CIF tree: A data structure for hierarchical in-line algorithms*, in Proceedings of the ACM/IEEE 19th Annual Design Automation Conference, Las Vegas, NV, June 1982, pp. 352–257.
- [26] A. KLINGER, *Patterns and search statistics*, in Optimizing Methods in Statistics, J. S. Rustagi, ed., Academic Press, New York, 1971, pp. 303–337.
- [27] K. KNOWLTON, *Progressive transmission of grey-scale and binary pictures by simple, efficient, and lossless encoding schemes*, Proc. IEEE 68 (1980), pp. 885–896.
- [28] D. E. KNUTH, *The Art of Computer Programming: Sorting and Searching*, vol. 3, 2nd ed., Addison-Wesley, Reading, MA, 1998.
- [29] J. H. LEE, D. H. KIM, AND C. W. CHUNG, *Multi-dimensional selectivity estimation using compressed histogram information*, in Proceedings of the ACM SIGMOD Conference, Philadelphia, PA, June 1999, pp. 205–214.
- [30] C. MATHIEU, C. PUECH, AND H. YAHIA, *Average efficiency of data structures for binary image processing*, Inform. Process. Lett., 26 (1987), pp. 89–93.
- [31] Y. MATIAS, J. S. VITTER, AND M. WANG, *Wavelet-based histograms for selectivity estimation*, in Proceedings of the ACM SIGMOD Conference, Seattle, WA, June 1998, pp. 448–459.

- [32] D. MEAGHER, *Geometric modeling using octree encoding*, Comput. Graphics Image Process., 19 (1982), pp. 129–147.
- [33] B. MOON AND J. H. SALTZ, *Scalability analysis of declustering methods for multidimensional range queries*, IEEE Trans. Knowledge Data Engrg., 10 (1998), pp. 310–327.
- [34] M. MURALIKRISHNA AND D. J. DEWITT, *Equi-depth histograms for estimating selectivity factors for multi-dimensional queries*, in Proceedings of the ACM SIGMOD Conference, Chicago, IL, June 1988, pp. 28–36.
- [35] R. C. NELSON AND H. SAMET, *A consistent hierarchical representation for vector data*, Comput. Graphics, 20 (1986), pp. 197–206.
- [36] R. C. NELSON AND H. SAMET, *A Population Analysis of Quadtrees with Variable Node Size*, Tech. report TR-1740, Department of Computer Science, University of Maryland, College Park, MD, December 1986.
- [37] R. C. NELSON AND H. SAMET, *A population analysis for hierarchical data structures*, in Proceedings of the ACM SIGMOD Conference, San Francisco, CA, May 1987, pp. 270–277.
- [38] J. A. ORENSTEIN, *Spatial query processing in an object-oriented database system*, in Proceedings of the ACM SIGMOD Conference, Washington, DC, May 1986, pp. 326–336.
- [39] M. OUKSEL AND P. SCHEUERMANN, *Storage mappings for multidimensional linear dynamic hashing*, in Proceedings of the ACM SIGACT-SIGMOD Symposium on Principles of Database Systems (PODS), Atlanta, GA, March 1983, pp. 90–105.
- [40] B. U. PAGEL, H. W. SIX, H. TOBEN, AND P. WIDMAYER, *Towards an analysis of range query performance in spatial data structures*, in Proceedings of the 12th Annual ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS), Washington, DC, May 1993, pp. 214–221.
- [41] A. PAPOULIS, *Probability, Random Variables, and Stochastic Processes*, 3rd ed., McGraw Hill, New York, 1991.
- [42] C. PUECH AND H. YAHIA, *Quadtrees, octrees, hyperoctrees: A unified analytical approach to three data structures used in graphics, geometric modeling and image processing*, in Proceedings of the ACM Symposium on Computational Geometry, Baltimore, MD, June 1985, pp. 272–280.
- [43] H. SAMET, *The Design and Analysis of Spatial Data Structures*, Addison-Wesley, Reading, MA, 1990.
- [44] H. SAMET, *Applications of Spatial Data Structures: Computer Graphics, Image Processing, and GIS*, Addison-Wesley, Reading, MA, 1990.
- [45] H. SAMET, *Foundations of Multidimensional and Metric Data Structures*, Morgan Kaufmann, San Francisco, CA, 2006.
- [46] H. SAMET, A. ROSENFELD, C. A. SHAFFER, R. C. NELSON, AND Y. G. HUANG, *Application of hierarchical data structures to geographical information systems: Phase III*, Tech. report TR-1457, Department of Computer Science, University of Maryland, College Park, MD, 1984.
- [47] H. SAMET, C. A. SHAFFER, AND R. E. WEBBER, *Digitizing the plane with cells of non-uniform size*, Inform. Process. Lett., 24 (1987), pp. 369–375.
- [48] H. SAMET AND M. TAMMINEN, *Efficient component labeling of images of arbitrary dimension represented by linear bintrees*, IEEE Trans. Pattern Anal. Mach. Intell., 10 (1988), pp. 579–586.
- [49] H. SAMET AND R. E. WEBBER, *Storing a collection of polygons using quadtrees*, ACM Trans. Graphics, 4 (1985), pp. 182–222.
- [50] L. A. SANTALO, *Integral Geometry and Geometric Probability*, Encyclopedia of Math. Appl. 1, Addison-Wesley, Reading, MA, 1976.
- [51] C. A. SHAFFER, *A formula for computing the number of quadtree node fragments created by a shift*, Pattern Recognition Lett., 7 (1988), pp. 45–49.
- [52] C. A. SHAFFER, R. JUUVADI, AND L. S. HEATH, *Generalized comparison of quadtree and bintree storage requirements*, Image Vision Comput., 11 (1993), pp. 402–412.
- [53] C. A. SHAFFER, H. SAMET, AND R. C. NELSON, *QUILT: A geographic information system based on quadtrees*, Int. J. Geographical Inform. Systems, 4 (1990), pp. 103–131.
- [54] M. TAMMINEN, *Encoding pixel trees*, Comput. Vision Graphics Image Process., 28, (1984), pp. 44–57.
- [55] M. TAMMINEN, *Comment on quad- and octrees*, Comm. ACM, 27 (1984), pp. 248–249.
- [56] M. VASSILAKOPOULOS AND Y. MANOLOPOULOS, *Analytical results on the quadtree storage requirements*, in Proceedings of the 5th International Conference on Computer Analysis of Images and Patterns (CAIP '93), D. Chetverikov and W. G. Kropatsch, eds., Lecture Notes Comput. Sci. 719, Springer-Verlag, Berlin, 1993, pp. 41–48.

HOLOGRAPHIC PROOFS AND DERANDOMIZATION*

DIETER VAN MELKEBEEK[†] AND RAHUL SANTHANAM[‡]

Abstract. We derive a stronger consequence of EXP (deterministic exponential time) having polynomial-size circuits than was known previously, namely that for each language $L \in \mathcal{P}$ (polynomial time), and for each efficiently decidable error-correcting code E having nontrivial relative distance, there is a simulation of L in Merlin-Arthur polylogarithmic time that fools all deterministic polynomial-time adversaries for inputs that are codewords of E . Using the connection between circuit lower bounds and derandomization, we obtain uniform assumptions for derandomizing BPP (probabilistic polynomial time). Our results strengthen the space-randomness tradeoffs of Sipser [*J. Comput. System Sci.*, 36 (1988), pp. 379–383], Nisan and Wigderson [*J. Comput. System Sci.*, 49 (1994), pp. 149–167], and Lu [*Comput. Complexity*, 10 (2001), pp. 247–259].

We also consider a more quantitative notion of simulation, where the measure of success of the simulation is the fraction of inputs of a given length on which the simulation works. Among other results, we show that if there is no polynomial-time bound t such that \mathcal{P} can be simulated well by Merlin-Arthur machines operating in time t , then for any $\epsilon > 0$ there is a simulation of BPP in \mathcal{P} that works for all but 2^{n^ϵ} inputs of length n . This is a uniform strengthening of a recent result of Goldreich and Wigderson [*Proceedings of the 6th International Workshop on Randomization and Approximation Techniques in Computer Science*, 2002, pp. 209–223].

Finally, we give an unconditional simulation of multitape Turing machines operating in probabilistic time t by Turing machines operating in deterministic time $o(2^t)$. We show similar results for randomized NC^1 circuits.

Our proofs are based on a combination of techniques in the theory of derandomization with results on holographic proofs.

Key words. holographic proofs, derandomization, Arthur-Merlin games

AMS subject classifications. 68Q15, 68Q17, 68Q10, 03D15

DOI. 10.1137/S0097539703438629

1. Introduction. The power of randomness is a central issue in the theory of computing. Although the use of random bits seems to simplify a lot of computational tasks, their actual computational power remains open. We have techniques that allow us to eliminate the randomness from many important randomized algorithms at a small cost. Just recently, a deterministic polynomial-time algorithm was discovered for the “prime” example of a computational problem where randomness seemed to provide a superpolynomial speed-up [AKS02]. For arbitrary (bounded-error) randomized computations running in time t , we have strong indications that they can be simulated deterministically in time $t^{O(1)}$. However, we have not even been able to prove the existence of a deterministic simulation that runs in time $o(2^t)$.

The standard approach for deterministic simulations uses the notion of a pseudorandom generator: an efficient deterministic procedure that expands short random seeds into longer strings such that no small circuit can distinguish the outputs of

*Received by the editors December 15, 2003; accepted for publication (in revised form) February 3, 2005; published electronically September 8, 2005. An extended abstract of this paper appeared in the *Proceedings of the 18th Annual IEEE Conference on Computational Complexity* [SvM03]

<http://www.siam.org/journals/sicomp/35-1/43862.html>

[†]Department of Computer Sciences, University of Wisconsin–Madison, Madison, WI 53706 (dieter@cs.wisc.edu). The work of this author was partially supported by NSF Career award CCR-0133693.

[‡]Department of Computer Science, University of Chicago, Chicago, IL 60637 (rahul@cs.uchicago.edu). The work of this author was partially supported by NSF Career award CCR-0133693 during a visit at the University of Wisconsin–Madison.

the generator from truly random strings. Given a pseudorandom generator G that “fools” circuits of size t , and a bounded-error randomized algorithm A that can be implemented by circuits of size t , when A is run with $G(\sigma)$ on its random tape for seed σ chosen uniformly at random, the acceptance probability of A is approximately the same as when it is run with truly random bits. Thus, we obtain a deterministic simulation of A on a given input x by cycling through all short seeds σ , running A on x using $G(\sigma)$ as the random bit sequence, and outputting the majority result of these runs.

As a result of a long line of research (e.g., [Yao82, BM84, NW94, BFNW93, IW97]), we know that pseudorandom generators exist iff exponential time requires large circuits. Thus, efficient deterministic simulations of arbitrary randomized computations exist unless exponential time has small circuits. Recent work [IKW02, KI03] indicates that circuit lower bounds are also necessary for the existence of such simulations.

The connections between deterministic simulations and circuit lower bounds are a double-edged sword. On the one hand, the connections between deterministic simulations and circuit lower bounds provide strong evidence that the randomness in probabilistic polynomial-time algorithms can be eliminated, since exponential time is widely believed to require large circuits. On the other hand, proving circuit lower bounds for exponential time has been one of the most elusive pursuits in the theory of computing. A justified attempt to circumvent that problem is to relax the notion of simulation. Instead of requiring that the simulation be correct on *every* input, we may content ourselves with simulations that err on few inputs, or for which it is computationally hard to pinpoint an error [Kab01]. For example, we may be happy if no probabilistic polynomial-time machine succeeds with significant probability in generating, on input 0^n , for infinitely many n , an input of length n on which the simulation fails.¹ Borrowing terminology from cryptography, we then say that the simulation “fools” probabilistic polynomial-time adversaries. By allowing machines other than probabilistic polynomial-time ones, we obtain other notions of simulations against uniform adversaries; see section 2 for the precise definitions. The standard notion of simulation coincides with the relaxed one for nonuniform adversaries, i.e., polynomial-size circuits instead of polynomial-time probabilistic machines.

We apply these relaxed notions of simulations to inclusions of deterministic and nondeterministic time in small Merlin-Arthur classes [BM88]. In fact, we consider four types of simulations of various types of computations: (i) the traditional one against nonuniform adversaries, (ii) the recent one against uniform adversaries, (iii) a slightly stronger type of simulation than (ii) in which the adversary only needs to output a list of inputs (instead of a single input) such that the simulation fails on at least one of these inputs, and (iv) a more quantitative notion. The last notion uses the fraction of inputs on which the simulation works as the measure of success [GW02]. Whereas a simulation against a uniform adversary may err on a substantial fraction of the inputs, but these inputs are hard to generate, the quantitative type of simulation is guaranteed to work on a large fraction of the inputs, but inputs on which it errs may be easy to generate.

We establish new tradeoffs between deterministic simulations of randomized computations and very efficient Merlin-Arthur simulations of deterministic and nondeterministic time. We also prove an unconditional deterministic time $o(2^t)$ simulation of randomized computations running in time t in the multitape Turing machine model. Our results make use of the known pseudorandom generator constructions based on

¹This has been proposed as an interesting formalization of the notion of a “heuristic” [IW01].

hard functions, and of the results on holographic proofs (also known as probabilistically checkable proofs).

1.1. Simulations against uniform adversaries. Recall that efficient deterministic simulations of arbitrary randomized computations against nonuniform adversaries exist unless exponential time has small circuits. We investigate simulations of deterministic and nondeterministic time against uniform adversaries that follow from the assumption that exponential time has small circuits. Thus, we obtain tradeoffs between simulations of randomized computations against nonuniform adversaries, and simulations of deterministic and nondeterministic computations against uniform adversaries.

Following earlier work [KL82], Nisan (credited in [BFL91]) showed that if exponential time has small circuits, then it collapses to MA (Merlin-Arthur polynomial time). We show a stronger consequence by considering simulations of problems in P (deterministic polynomial time). We need to settle for simulations of promise problems in P, where the promise is that the input is a codeword of a “nice” code E , namely a polynomial-time decidable error-correcting code with relative distance at least an inverse polylogarithmic fraction of the length of the codeword.

We first give the precise statement of our result and then explain our notation.

THEOREM 1. *If $\text{EXP} \subseteq \text{SIZE}(\text{poly})$ (where EXP refers to deterministic exponential time), then for each language $L \in \text{P}$ and each nice code E , $(E, L) \in \text{quasi}_{\text{P}}\text{-MAPOLYLOG}$.*

The consequent in the statement of Theorem 1 requires some explanation. (E, L) is the promise problem consisting of the decision problem L restricted to inputs in E . MAPOLYLOG is the “scaled-down” version of MA, where the verifier has random access to the input and runs in polylogarithmic time. The “*quasi*” refers to the stronger type of simulation against an adversary that runs in P (type (iii) above). More precisely, “ $(E, L) \in \text{quasi}_{\text{P}}\text{-MAPOLYLOG}$ ” means that there exists a machine M for which no deterministic polynomial-time adversary can generate a list of inputs such that M does not behave like a Merlin-Arthur polylogarithmic time machine for L on all inputs in the list which belong to E . The inclusion $\text{EXP} \subseteq \text{MA}$ follows from the consequent of Theorem 1 and the existence of a polynomial-time computable linear nice code E by a translation argument. Thus, Theorem 1 gives a stronger consequence of the assumption that EXP has polynomial-size circuits than was known before. Similar theorems hold for PSPACE and $\text{P}^{\#\text{P}}$.

In order to prove Theorem 1, we cast Nisan’s argument as an application of the “easy witness” method [Kab01, IKW02] to holographic proofs, i.e., as a search for holographic proofs that can be described by small circuits.

Given Theorem 1, the known connections between circuit lower bounds and derandomization imply tradeoffs between efficient simulations of P that fool uniform adversaries and deterministic simulations of BPP (probabilistic polynomial time) for a range of parameters. In a seminal paper, Sipser [Sip88] showed tradeoffs between efficient simulations of P in deterministic space and derandomizations of RP (randomized polynomial time with one-sided error)—indeed, the easy witness method is implicit in [Sip88]. These results were extended by Nisan and Wigderson [NW94] and by Lu [Lu01]. Using Theorem 1, we further strengthen these tradeoffs. For example, we obtain the following corollary to Theorem 1.

COROLLARY 2. *For each language $L \in \text{P}$ and nice code E , $(E, L) \in \text{quasi}_{\text{P}}\text{-MAPOLYLOG}$, or $\text{BPP} \subseteq \text{infinitely often (i.o.) SUBEXP}$.*

Theorem 1 provides an interesting perspective on proving circuit lower bounds

for EXP. We observe that PARITY is not contained in MAPOLYLOG unconditionally. Thus P is not contained in MAPOLYLOG. Theorem 1 says that a stronger separation of P and MAPOLYLOG implies circuit lower bounds for EXP.² This opens the possibility that methods from low-level circuit complexity may be useful in this area. In fact, we apply the circuit lower bounds of Hastad for PARITY [Hås86] in a novel way to obtain a partial converse to one of our results. See section 3 for more details.

We also show tradeoffs between simulations of NP in MAPOLYLOG that fool nondeterministic adversaries, and nondeterministic simulations of MA. For technical reasons explained in section 4, we cannot use our stronger notion of simulation against uniform adversaries here but need to switch back to the weaker notion, where the adversary outputs a single output of a given length on which the simulation is supposed to fail. Among other results, we also give a different proof of the gap theorem due to Impagliazzo, Kabanets, and Wigderson [IKW02] that either $\text{NEXP} = \text{MA}$ or else $\text{MA} \subseteq i.o. [\text{NTIME}[2^{n^\epsilon}]/n^\epsilon]$ for every $\epsilon > 0$.

1.2. Quantitative simulations. A second family of results in our paper concerns a different, more quantitative notion of simulation, where the measure of success is the fraction of inputs of a given length on which the simulation works. This relates to recent work of Goldreich and Wigderson [GW02], who obtained efficient deterministic simulations of BPP that work on almost all inputs under a certain nonuniform assumption (see section 5 for the precise assumption). Their idea, in a nutshell, is to extract randomness from the *input*—most inputs are “random” enough that this can be done. By combining the easy witness method and results on holographic proofs with the idea of Goldreich and Wigderson, we derive a stronger and uniform version of their result, as follows.

THEOREM 3. *If there is no polynomially bounded t such that $\text{P} \subseteq [\text{io} - \text{corr}_{2/3}]\text{-MATIME}(t)$, then for every $\epsilon > 0$, $\text{BPP} \subseteq \text{corr}_{1-2^{n^\epsilon}/2^n}\text{P}$.*

This theorem states that if there is no fixed polynomial t such that for every language L in P there is a machine M which is a Merlin-Arthur machine for L operating in time t on at least a fraction $2/3$ of the inputs for infinitely many input lengths, then for every $\epsilon > 0$ and every language L in BPP there is a simulation of L in P which succeeds on all but 2^{n^ϵ} inputs for all input lengths.

1.3. Unconditional simulations. The standard approach to derandomization in the time-bounded setting is the construction of an efficient pseudorandom generator that fools all computations that run within time t on the model of computation under consideration. It is well known that the existence of such a pseudorandom generator $g_n : \{0, 1\}^\sigma \rightarrow \{0, 1\}^n$ (where $\sigma < n$) implies the existence of an explicit problem outside a nonuniform version of the class \mathcal{C} being derandomized, namely the problem of deciding whether a string of length $\sigma+1$ is the initial segment of a string in the range of g_n . If that problem were in the class \mathcal{C}/n , running the decision procedure with the correct advice on a random input of length $\sigma+1$ would accept with probability at most $1/2$, whereas the simulated process using the pseudorandom generator would accept with probability 1. Thus, short of establishing new lower bounds for explicit functions, one can only hope to construct pseudorandom generators that are unconditionally secure for settings where such bounds are known.

In general models of computation, only very weak lower bounds of the above type are known. On one-tape Turing machines, the strongest known is a quadratic lower

²[AG91] also considers the consequences of strong separations of P from lower-level classes. The notion of a “strong separation” there is different from ours.

bound for deciding the palindromes. Almost matching that lower bound, [INW94] managed to construct a pseudorandom generator with seed length $\sigma = \tilde{O}(\sqrt{t})$ for probabilistic one-tape Turing machine computations that run in time t . The approach of [INW94] does not yield any nontrivial results for multitape Turing machines with more than one tape. For a fixed number $k > 1$ of tapes and a fixed alphabet, the best lower bounds known are of the form cn for some constant $c > 1$, so one may hope to obtain a pseudorandom generator with seed length $\sigma = (1 - \delta)t$, where δ is a positive constant depending on the number of tapes and the alphabet size. We manage to do so. The straightforward deterministic simulation using our pseudorandom generator yields the following result.

THEOREM 4. *For any integers $k > 0$ and $\beta > 0$, there is a constant $\delta > 0$ such that for each language L accepted by a bounded-error probabilistic k -tape Turing machine with alphabet size β running in time t , $L \in \text{DTIME}(2^{(1-\delta)t})$.*

To the best of our knowledge, the fact that time t probabilistic multitape Turing machine computations can be simulated deterministically in time $o(2^t)$ is new. We also demonstrate our technique for randomized NC^1 computations.

Note that the typical constructions of pseudorandom generators [NW94, SU01] are unable to establish Theorem 4 because these constructions lose a superlinear factor in the transformation of a hard function into a pseudorandom generator, and we only have linear lower bounds to start from. Thus, we need a different angle. Our approach is inspired by the idea of Impagliazzo and Zuckerman [IZ89] to recycle random bits, and specifically by an application of this idea due to Nisan and Zuckerman [NZ96]. The latter authors considered randomized space s computations, and argued as follows. Consider the configuration C reached after processing the first $\Theta(s)$ random bits r . Since there are only $2^{O(s)}$ possible configurations C in total, we can ignore the contributions of those C 's that have probability less than $2^{-\Omega(s)}$ of occurring. For every other C , the distribution of r conditioned on reaching C has $\Omega(s)$ bits of randomness, which we can extract and feed to the randomized algorithm as the next $\Omega(s)$ (pseudo-)random bits. This way, we save a number of random bits equal to $\Omega(s)$ minus the seed length of the extractor.

The small number of reachable configurations plays a crucial role in the correctness argument. Reaching the same configuration C guarantees that the future behavior of the original machine will be the same, irrespective of the actual value of r . In our setting, we do not have strong bounds on the number of reachable configurations. However, we can bound the number of different future behaviors of a Turing machine during the next s steps. Since a Turing machine can move each of its k tape heads over at most one position per step, only $2ks$ tape cells can affect the next s steps. Thus, the number of different behaviors is bounded by $O(\alpha^{2ks})$, where α is the alphabet size of the machine. Using this bound in the argument of [NZ96] and an appropriate extractor, we obtain a simulation that runs in time polynomial in t and uses only $(1 - \delta)t$ random bits for some positive constant δ .

1.4. Organization. The rest of this paper is organized as follows. We define our main concepts in section 2. We discuss our simulations against uniform adversaries for deterministic time in section 3, and then for nondeterministic time in section 4. Next we consider the quantitative simulations in section 5, and we end with our unconditional simulation results in section 6.

2. Preliminaries. In this section, we describe the complexity classes we consider, and formally define the various types of simulation between them. We review results on pseudorandom generators and extractors, as well as on holographic proofs,

and explain how the latter will be used in combination with “nice” codes in the rest of the paper.

We will use numbered definitions for notions that originate in this paper or are not widely known, and unnumbered definitions for more conventional notions.

2.1. Complexity classes. The definitions of standard deterministic, nondeterministic, and probabilistic resource-bounded classes can be found in [BDG88] and [BDG90]. We will also need definitions of sublinear time complexity classes. These classes are defined by Turing machines with a separate index tape on which the address of an input bit can be written. Given this address, the machine can access the bit in constant time. Analogous to classes defined by polynomial-time machines, we can define DPOLYLOG, NPOLYLOG, BPPOLYLOG, and MAPOLYLOG. All of our results hold for both the one-sided and two-sided error version of MAPOLYLOG. In proofs, we always choose the model that makes our results stronger. We can also define versions of the classes above where the machine takes advice of a given length—we assume that the advice is written on a separate tape and that the machine has random access to the advice. If \mathcal{C} is a complexity class defined by machines of a certain kind, $\mathcal{C}/poly$ denotes the complexity class defined by machines of the right kind taking polynomial length advice strings, and $\mathcal{C}/qpoly$ denotes the complexity class defined by machines of the right kind taking quasi-polynomial (i.e., $2^{O(polylog(n))}$) length advice strings (where the complexity is measured in terms of the original input size).

We use standard definitions of nonuniform classes. The class of languages with circuits of size t is denoted by $\text{SIZE}(t)$, and the class of languages that have \mathcal{A} -oracle circuits of size t for some complexity class \mathcal{A} is denoted $\text{SIZE}^{\mathcal{A}}(t)$.

If L is a language and \mathcal{C} is a class, we say $L \in i.o.\mathcal{C}$ if there is a language $L' \in \mathcal{C}$ such that $\{n : L \cap \{0, 1\}^n = L' \cap \{0, 1\}^n\}$ is infinite.

In general, complexity classes are defined by acceptance and rejection criteria on the behavior of a specific kind of computation device on an input. The acceptance and rejection criteria are always exclusive but need not be exhaustive. Classes for which they are not exhaustive, such as BPP and MA, are referred to as *promise classes*. We introduce some notation to capture the general case in which the criteria may or may not be exhaustive.

DEFINITION 5. *Given a complexity class \mathcal{C} , a computation device M of the right kind, and an input x , we say that M conforms to \mathcal{C} on input x if the disjunction of the acceptance and rejection criteria for \mathcal{C} holds for the computation of M on input x .*

A *promise problem* is a formalization of the notion of a partially defined decision problem. Formally, a promise problem is a pair (X, L) , where $X, L \subseteq \{0, 1\}^*$. X denotes the domain of the problem (the set of inputs that satisfy the “promise” of the problem), and L denotes an underlying (fully defined) decision problem. We now define what it means for a promise problem to belong to a complexity class.

DEFINITION 6. *Given a complexity class \mathcal{C} , we write $(X, L) \in \mathcal{C}$ if there exists a computation device M of the right kind such that for all $x \in X$, M conforms to \mathcal{C} on input x , and M accepts iff $x \in L$.*

Note that the behavior of M on inputs $x \notin X$ may be arbitrary.

2.2. Simulations. In this paper, we are concerned with various simulations of one type of computation by another type of computation. According to the conventional notion, a simulation is successful if it fools all nonuniform adversaries, i.e., the languages accepted by the two machines coincide on all but finitely many inputs. Kabanets [Kab01] formalized a notion of what it means for a simulation to fool uniform

adversaries (which is weaker than the conventional notion of a simulation). He considered a simulation of a language L in a class \mathcal{C} against adversaries in a class \mathcal{A} to correspond to a language $L' \in \mathcal{C}$ such that no adversary in \mathcal{A} can efficiently find an input on which the simulation makes an error, i.e., an input on which L and L' differ, for infinitely many input lengths. We need to refine the notion of simulation against uniform adversaries for the case of simulation by promise classes. In general, we consider a simulation to correspond to a specific *machine* rather than to a *language*, and say that the simulation makes an error on an input if either the simulating machine does not conform to the simulating class on the input, or the simulating machine and the simulated language do not agree on the input. Arguably, our notion of simulation against uniform adversaries is more natural (though weaker) than the earlier notion in the case of simulations by promise classes. We point out that this is the first paper that considers the issues involved in such simulations, and that our notion coincides with the earlier one in the case of simulations by nonpromise classes.

DEFINITION 7. *Given a complexity class \mathcal{C} , a machine M of the right kind, a language L , and an input x , M makes a simulation error on input x for the simulation of L in \mathcal{C} if one of the following two conditions holds:*

1. M does not conform to \mathcal{C} on input x , or
2. M accepts x iff $x \notin L$.

Next, we formally define a notion of “uniform adversary.”

DEFINITION 8. *A refuter is a deterministic Turing machine that, on input 0^n , outputs a string of length n .*

P and SUBEXP are the classes of refuters operating in polynomial time and subexponential time, respectively, and Tally is the singleton class consisting of the refuter that outputs 0^n on input 0^n . LOGSPACE is the class of refuters equipped with a two-way read-only input tape, a one-way write-only output tape, and a logarithmic amount of work space.

Next, we define what it means for a refuter to succeed against a simulation.

DEFINITION 9. *A refuter N succeeds at length n against a simulation M of a language L in a complexity class \mathcal{C} if M makes a simulation error on input $N(0^n)$ for the simulation of L in \mathcal{C} .*

DEFINITION 10. *A refuter N succeeds against a simulation M of L in \mathcal{C} if N succeeds at length n against M for infinitely many n .*

We say that a simulation M of L fools a refuter N if N does not succeed against M . We define what it means to have simulations in a complexity class that fool refuters from a given class.

DEFINITION 11. *Given a complexity class \mathcal{C} and a class \mathcal{A} of refuters, $\text{pseudo}_{\mathcal{A}}\text{-}\mathcal{C}$ is the class of languages L such that there is a simulation M of L in \mathcal{C} that fools every refuter in \mathcal{A} .*

We reiterate that a successful simulation is a *single* machine that fools *every* refuter in a class.

We introduce a weaker concept of refutation.

DEFINITION 12. *A weak refuter is a deterministic Turing machine that, on input 0^n , outputs a set of strings, each of which is of length n .*

Note that the resource bounds of the weak refuter implicitly define a limit on the cardinality of the set of strings.

DEFINITION 13. *A weak refuter N succeeds at length n against a simulation M of a language L in a complexity class \mathcal{C} if there is at least one $x \in N(0^n)$ such that M makes a simulation error on input x for the simulation of L in \mathcal{C} .*

The notion of success of a weak refuter and the notion of a simulation fooling a weak refuter are defined analogously to the corresponding notions for refuters.

DEFINITION 14. *Given a complexity class \mathcal{C} and a class \mathcal{A} of weak refuters, $quasi_{\mathcal{A}}\text{-}\mathcal{C}$ is the class of languages L such that there is a simulation M of L in \mathcal{C} that fools every weak refuter in \mathcal{A} .*

Clearly, for each \mathcal{A} and \mathcal{C} , $\mathcal{C} \subseteq quasi_{\mathcal{A}}\text{-}\mathcal{C} \subseteq pseudo_{\mathcal{A}}\text{-}\mathcal{C}$.

We can modify and extend the notions of “refuter” and “weak refuter” in various ways. We can redefine the notion of success of a refuter to mean that a refuter succeeds at length n for almost every n rather than for infinitely many n .

DEFINITION 15. *A refuter (resp., a weak refuter) N succeeds almost everywhere against a simulation M of L in \mathcal{C} if N succeeds at length n against M for all but finitely many n .*

DEFINITION 16. *$[io\text{-}pseudo_{\mathcal{A}}]\text{-}\mathcal{C}$ (resp., $[io\text{-}quasi_{\mathcal{A}}]\text{-}\mathcal{C}$) is the class of languages L such that there is a simulation M of L in class \mathcal{C} for which no refuter (resp., weak refuter) in \mathcal{A} succeeds almost everywhere against the simulation.*

We can also consider probabilistic and nondeterministic refuters, which can output different strings on different computation paths.

DEFINITION 17. *A probabilistic refuter is a probabilistic Turing machine, which, on input 0^n , for each computation path outputs a string of length n .*

DEFINITION 18. *A probabilistic refuter succeeds at length n against a simulation M of L in \mathcal{C} if, with probability at least $2/3$ over the random bits of the refuter, M makes a simulation error on the string output by the refuter for the simulation of L in \mathcal{C} .*

DEFINITION 19. *A nondeterministic refuter is a nondeterministic Turing machine, which, on input 0^n , outputs a string of length n on each computation path and accepts on at least one computation path.*

DEFINITION 20. *A nondeterministic refuter succeeds at length n against a simulation M of L in \mathcal{C} if, for all strings x produced by the refuter on accepting paths on input 0^n , M makes a simulation error on x for the simulation of L in \mathcal{C} .*

Probabilistic weak refuters have the same relation to probabilistic refuters as deterministic weak refuters have to deterministic refuters.

Space-bounded refuters and weak refuters are equipped with a separate output tape on which a string or list of strings is written. The output tape does not contribute to the space usage.

The above notions generalize naturally to simulations of promise problems against uniform adversaries.

DEFINITION 21. *A machine M makes a simulation error on input x for the simulation of the promise problem (X, L) in complexity class \mathcal{C} if $x \in X$ and either M does not conform to \mathcal{C} on x or M does not agree with L on x .*

The definitions of “ $(X, L) \in quasi_{\mathcal{A}}\text{-}\mathcal{C}$ ” and “ $(X, L) \in pseudo_{\mathcal{A}}\text{-}\mathcal{C}$ ” are the same as for the underlying decision problem modulo the modified notion of simulation error.

We also define a quantitative notion of simulation, where the measure of success is the fraction of inputs of a given length on which the simulation works.

DEFINITION 22. *Given a language L and a complexity class \mathcal{C} , $L \in corr_s\text{-}\mathcal{C}$ if there is a machine M such that for each input length n , for at least an s fraction of inputs x of length n , M conforms to \mathcal{C} on x and agrees with L on x .*

DEFINITION 23. *Given a language L and a complexity class \mathcal{C} , $L \in [io\text{-}corr]_s\text{-}\mathcal{C}$ if there is a machine M such that for infinitely many input lengths n , for at least an s fraction of inputs x of length n , M conforms to \mathcal{C} on x and agrees with L on x .*

2.3. Pseudorandom generators and extractors. A pseudorandom generator G is a sequence of functions g_n , $n \geq 1$, where g_n maps $\{0, 1\}^{\sigma(n)}$ to $\{0, 1\}^n$ for some $\sigma(n) < n$, such that for all circuits C of size less than n ,

$$\left| \Pr_x[C(x) = 1] - \Pr_y[C(g_n(y)) = 1] \right| < \frac{1}{n},$$

where x is uniformly distributed over $\{0, 1\}^n$ and y is uniformly distributed over $\{0, 1\}^{\sigma(n)}$. The pseudorandom generators we consider will be computable in time $2^{O(\sigma(n))}$. We will sometimes abuse notation and call g_n a pseudorandom generator for some output length n if g_n fools all circuits of size less than n . We refer the reader to the survey paper by Goldreich [Gol00] for more background on pseudorandom generators.

The following theorem of Impagliazzo and Wigderson shows how to obtain a pseudorandom generator with logarithmic seed length from the truth table of a hard function.

THEOREM 24 (see [IW97]). *For every $\epsilon > 0$, there are constants c and d and a polynomial-time procedure, which, given as input the truth table of a Boolean function f on $\log(n)$ inputs, computes the graph of a function $g_{n^{\epsilon/c}} : \{0, 1\}^{d \log(n)} \rightarrow \{0, 1\}^{n^{\epsilon/c}}$, such that when f does not have circuits of size n^ϵ , $g_{n^{\epsilon/c}}$ is a pseudorandom generator.*

Note that the truth table of a language in \mathbf{E} (linear exponential time) can be generated in time polynomial in the size of the truth table (just by enumerating inputs of a given length and checking, for each input, whether the input is in the language or not). Thus the generator g corresponding to $f \in \mathbf{E}$ in Theorem 24 is computable in time $\text{poly}(n)$.

Impagliazzo and Wigderson also showed a tradeoff between uniform simulations of \mathbf{EXP} and efficient simulations of \mathbf{BPP} , which can be expressed in our notation as follows.

THEOREM 25 (see [IW01]). *If $\mathbf{EXP} \neq \mathbf{BPP}$, then $\mathbf{BPP} \subseteq [\text{io} - \text{pseudo}_{\mathbf{BPP}}]\text{-SUBEXP}$.*

Indeed, their argument actually gives the stronger consequence that $\mathbf{BPP} \subseteq [\text{io} - \text{quasi}_{\mathbf{BPP}}]\text{-SUBEXP}$.

For our results in section 6, we need to define the notion of an extractor. First we give some notation and definitions concerning probability distributions. Let D be a probability distribution on a finite set S . We write $r \leftarrow D$ if r is a random variable distributed according to D . Given $x \in S$ and $A \subseteq S$, $D(x)$ is the probability weight assigned to element x of S by the distribution D , and $D(A)$ is the probability that the event A occurs. The *min-entropy* $H_\infty(D)$ of the distribution D is defined to be $\min_{x \in S} \log(1/D(x))$. Given two distributions D_1 and D_2 over the same set S , the *statistical difference* between D_1 and D_2 is defined to be $\max_{A \subseteq S} |D_1(A) - D_2(A)|$. D_1 and D_2 are said to be ϵ -close for $\epsilon \geq 0$ if the statistical difference between D_1 and $D_2 \leq \epsilon$. If D_1 and D_2 are ϵ -close distributions on a set S , and f is a deterministic function with domain S , the distributions $f(D_1)$ and $f(D_2)$ are also ϵ -close. We let U_n denote the uniform distribution on the set $\{0, 1\}^n$.

An extractor is a function that transforms a distribution with high min-entropy to one that is close to uniform, given a short random seed. More formally, a (k, ϵ) extractor is a function $\text{Ext} : \{0, 1\}^n \times \{0, 1\}^d \rightarrow \{0, 1\}^m$ such that for every distribution D on $\{0, 1\}^n$ with $H_\infty(D) \geq k$, the distribution $\text{Ext}(D, U_d)$ is ϵ -close to U_m . There has been a lot of work done on explicit constructions of “good” extractors, where a good extractor is one with small seed length (ideally $d = \log(n - k) + 2 \log 1/\epsilon + O(1)$)

such that almost all the min-entropy in the original distribution is converted to near-uniform random bits (ideally $m = k + d - 2 \log 1/\epsilon - O(1)$). We refer to the excellent paper by Shaltiel [Sha02] for a survey of recent extractor constructions. We will use the following extractor family, which has suboptimal seed length but extracts all the min-entropy of the source and is very efficiently computable.

THEOREM 26 (see [HR00]). *For each constant $\epsilon > 0$ and all functions $k(n)$ such that $0 \leq k(n) \leq n$, there is a family of $(k(n), \epsilon)$ extractors $Ext_k : \{0, 1\}^n \times \{0, 1\}^{O((\log(n))^3)} \rightarrow \{0, 1\}^{k(n)}$ that is uniformly computable in NC^1 .*

2.4. Proofs. We need to define what it means for a promise problem to have efficiently verifiable proofs.

DEFINITION 27. *Given a promise problem (X, L) and a complexity class \mathcal{C} , (X, L) is said to have proofs of size $p(n)$ verifiable in \mathcal{C} if there is a machine V (the verifier for L) such that the following conditions hold for all $x \in X$ (considering resource bounds for V as functions of $|x|$ rather than $|x| + p(|x|)$):*

1. *Completeness: If $x \in L$, there is a string $P(x)$ (the proof for x) of length $p(|x|)$ such that V conforms to \mathcal{C} on input $\langle x, P(x) \rangle$ and accepts $\langle x, P(x) \rangle$.*
2. *Soundness: If $x \notin L$, then for all strings y of length $p(|x|)$, V conforms to \mathcal{C} on input $\langle x, y \rangle$ and rejects $\langle x, y \rangle$.*

DEFINITION 28. *A complexity class \mathcal{B} is said to have proofs of size $p(n)$ verifiable in \mathcal{C} if for every language $L \in \mathcal{B}$, $(\{0, 1\}^*, L)$ has proofs of size $p(n)$ verifiable in \mathcal{C} .*

Holographic proofs are proofs for which there is a very efficient probabilistic verifier. Babai, Fortnow, and Lund [BFL91] showed that EXP has holographic proofs. More precisely, every language L in EXP has proofs of exponential size verifiable in coRP such that for $x \in L$ a proof for x can be computed from x in exponential time. In our applications, we require a “scaled-down” version of this result due to Babai et al. [BFLS91]. The natural way to scale down the result in [BFL91] would be to show that every language L in P has proofs of polynomial size verifiable in coRPOLYLOG such that for $x \in L$ a proof for x can be computed from x in polynomial time. However, such proofs cannot exist since a coRPOLYLOG verifier cannot read the entire input and determine that the purported proof is indeed a proof for that input. What we *can* do in randomized polylogarithmic time is verify a purported proof and, if it passes the verification, extract each bit of the input for which it is a proof with high probability.

The following theorem states that for each language decidable in deterministic time t , where t is polynomially bounded, there are proofs of size nearly linear in t with corresponding randomized polylogarithmic time proof verification and input extraction procedures. We note that for our main results we actually need only the existence of efficiently verifiable polynomial-sized proofs.

THEOREM 29 (see [BFLS91]). *For each language $L \in \text{DTIME}(t)$, where $t = \text{poly}(n)$, and all $\epsilon > 0$, there exist randomized $\text{polylog}(n)$ time procedures V and I and a deterministic $\text{poly}(n)$ time procedure P with the following properties:*

1. *For each $x \in L$ of length n , $y = P(x)$ is a string of length $t^{1+\epsilon}$ such that*
 - (a) $\Pr[V(y) \text{ accepts}] = 1$,
 - (b) $\Pr[I(y, i) = x_i] = 1$ for all $1 \leq i \leq n$.
2. *For each y of length $t^{1+\epsilon}$, if*

$$\Pr[V(y) \text{ accepts}] \geq \frac{1}{2},$$

then there exists a unique $x' \in L$ of length n such that

$$\Pr[I(y, i) = x'_i] \geq \frac{2}{3} \quad \text{for all } 1 \leq i \leq n.$$

The procedure V in the statement of the theorem corresponds to the verifier in the usual definition of a proof system, except that it has access only to the purported proof and not to the input x . The correspondence between the proof and the input is made via the auxiliary procedure I , which, given a proof y accepted with high probability, computes with high probability each bit of an input x' in the language. Note that it is not possible for a **coRPOLYLOG** procedure to check whether x' is the same as the actual input x , since x' and x may differ in very few bit positions, and there is not enough time for the procedure to check all bit positions. However, if x and x' were guaranteed to be codewords in a good error-correcting code, then a **coRPOLYLOG** procedure could just compare x and x' at a few randomly chosen bit positions to determine whether they were the same.

We define “nice codes” as codes with strong enough properties that the argument above can be made precise.

DEFINITION 30. *A nice code is an error-correcting code E such that*

1. *it is decidable in deterministic polynomial time whether a string is a codeword of E ;*
2. *E has relative distance bounded below by an inverse polylogarithmic function of the size of the codeword.*

Using Theorem 29, we can show the existence of efficient proof systems for promise problems in **P** where the promise is that the input is a codeword in a nice code.

THEOREM 31. *For each language $L \in \mathbf{P}$ and each nice code E , (E, L) has proofs of size $\text{poly}(n)$ verifiable in **coRPOLYLOG**. Furthermore, given $x \in L \cap E$, a proof that $x \in L$ is computable in time $\text{poly}(n)$.*

Proof. Let $L \in \mathbf{P}$. Define the language $L' = L \cap E$. (E, L) has proofs of size $\text{poly}(n)$ verifiable in **coRPOLYLOG** iff (E, L') has proofs of size $\text{poly}(n)$ verifiable in **coRPOLYLOG**; hence it suffices to make the argument for L' rather than L . Since E is a nice code, membership in E can be decided in polynomial time; hence $L' \in \mathbf{DTIME}(t)$ for some $t = \text{poly}(n)$. Fix an $\epsilon > 0$ and let V , I , and P be procedures as in the statement of Theorem 29 corresponding to L' . Let k be a constant such that any two different codewords of E of length m have relative distance at least $1/(\log(n))^k$. We define a verifier V' in **coRPOLYLOG** for (L', E) .

Given input $\langle x, y \rangle$ with $|x| = n$, V' runs in two phases. In the first phase, V' runs V on y . If V rejects, V' rejects. Otherwise, V' begins the second phase. It picks $O((\log(n))^{k+1})$ positions $i_1, \dots, i_{O((\log(n))^{k+1})}$ between 1 and n independently at random. For each position i_j , $1 \leq j \leq n$, V' runs $I(y, i_j)$ and checks whether the returned value is equal to x_{i_j} . If all the checks succeed, V' accepts; otherwise it rejects.

Clearly, V' runs in polylogarithmic time in the size of its input. We need to show that when $x \in L'$, there is some y of length $t^{1+\epsilon} = \text{poly}(n)$ such that V' accepts with probability 1 on $\langle x, y \rangle$, and for $x \in E \setminus L'$, V' rejects with probability at least $1/2$ on $\langle x, y \rangle$ for all y of length $t^{1+\epsilon}$.

The first case is easy. When $x \in L'$, condition 1 in the statement of Theorem 29 holds. Thus, V accepts on $P(x)$ with probability 1. Also, $I(P(x), i) = x_i$ with probability 1, and hence all the checks made by V' succeed with probability 1. It follows that V' accepts with probability 1 on input $\langle x, P(x) \rangle$.

For the second case, let $x \in E \setminus L'$. We assume there is a y of length $t^{1+\epsilon}$ such that V' accepts with probability at least $1/2$ on $\langle x, y \rangle$, and we derive a contradiction. Since $\langle x, y \rangle$ passes the first phase of V' , V accepts with probability at least $1/2$ on input y . This implies that there is a unique $x' \in L'$ such that I extracts each bit of x' from y independently with probability $\geq 2/3$. Since $L' \subseteq E$, $x' \in E$. Since $x \notin L'$ and $x' \in L'$, x and x' are different codewords of E , and hence they have relative distance at least $1/((\log(n))^k)$. V' rejects $\langle x, y \rangle$ in the second phase if at least one of the positions i on which x and x' differ is selected, and for that i the call $I(y, i)$ returns x'_i . The first condition occurs with probability close to 1 (by the lower bound on the relative distance of the codeword), and the second happens with probability at least $2/3$, which implies that V' rejects with probability greater than $1/2$. This is a contradiction. \square

It is well known that nice codes exist. In fact, the Justesen code [vL98] satisfies much stronger properties, which we will need for some of our results.

THEOREM 32. *There are positive constants r and δ such that there is a linear code $E : \{0, 1\}^{rn} \rightarrow \{0, 1\}^n$ that has relative distance at least δ and is computable and invertible in quasi-linear time as well as in logarithmic space.*

The existence of nice codes implies that Theorem 31 and the consequent of Theorem 1 are not vacuously true.

3. Simulations of deterministic time against uniform adversaries. In section 3.1, we prove our main result and show how it implies Nisan’s result. We also explore the question of whether the restriction to nice codes in the consequent of our main result can be eliminated. In section 3.2, we show variants of our main result in different ranges of the parameters, and prove a partial converse to one of our results using the idea of derandomizing probabilistic weak refuters. In section 3.3, we give tradeoffs between simulations of P in small Merlin-Arthur classes and deterministic simulations of BPP, which follow from our results and the connections between circuit lower bounds and derandomization given in [BFNW93] and [IW97]. Finally, in section 3.4, we state an analogue of our main result in the space-bounded setting.

3.1. Main result.

3.1.1. Proof of main result. In this subsection, we show that the hypothesis $\text{EXP} \subseteq \text{SIZE}(\text{poly})$ implies there exists a simulation of P in MAPOLYLOG that fools uniform polynomial-time adversaries that output codewords from a nice code. In order to do so, we cast Nisan’s argument that $\text{EXP} \subseteq \text{SIZE}(\text{poly})$ implies $\text{EXP} \subseteq \text{MA}$ as an application of the easy witness method to holographic proofs for exponential time.

Recall that a language L has holographic proofs if it has an efficient probabilistic verification system that needs to read only a small part of the proof for every sequence of random bits. EXP can be characterized as the class of languages with holographic proofs of exponential size that can be verified in probabilistic polynomial time and computed in exponential time [BFL91]. The easy witness method suggests trying proofs that are described by small circuits, i.e., proofs for which there exists a small circuit that outputs the i th bit of the proof on input i . We will refer to such proofs as “compressible.” This leads to the following Merlin-Arthur-type protocol for a language L in EXP : Given an input x , Merlin sends Arthur a small circuit describing a purported proof that $x \in L$. Arthur then runs the probabilistic verification procedure; each time that procedure needs a bit of the proof, Arthur evaluates the circuit on the corresponding input. By the properties of the holographic proof system, the only way

the protocol can fail if an input $x \in L$ does not have a compressible proof. If that happens only finitely often for every language L in EXP , we have that $\text{EXP} \subseteq \text{MA}$. If it happens infinitely often for some L in EXP , consider the concatenation of the holographic proofs for all inputs of a given length, and look upon that concatenation as the truth table of a Boolean function f . Since each proof can be computed in exponential time, we can generate the truth table of f for inputs of length n in time exponential in n . Since infinitely many proofs cannot be described by small circuits, f does not have small circuits. Thus, the function f is computable in exponential time but does not have small circuits, and thus $\text{EXP} \not\subseteq \text{SIZE}(\text{poly})$.

In order to scale down, we apply the easy witness method to holographic proofs for P . There are two issues in scaling down the above arguments. First, as explained in section 2.4 (see Theorem 31), the verifier is guaranteed to work correctly only on inputs that are codewords of a nice code E , not on all inputs of length n . This is why we consider simulations of *promise* problems corresponding to languages in P , where the promise is that the input belongs to E . Second, we cannot concatenate holographic proofs of *all* inputs of length n which are codewords of E , because the resulting string could be of exponential length, whereas we need one of polynomial length. This is where the weak refuter comes in: a polynomial-time algorithm that produces, on input 0^n , a list of strings of length n such that the Merlin-Arthur protocol fails on at least one of them. Concatenating the holographic proofs of the polynomially many inputs on the list yields the truth table we need. If, for every L in P , every refuter succeeds only finitely often, we have that $(E, L) \in \text{quasi}_{\text{P}}\text{-MAPOLYLOG}$ for all $L \in \text{P}$ by definition. Otherwise, we can generate in time $n^{O(1)}$ the truth table of a function on $O(\log n)$ inputs that does not have circuits of size $\log^{O(1)} n$. Thus $\text{E} \not\subseteq \text{SIZE}(\text{poly})$.

We now fill in the details.

Proof of Theorem 1. Let L be a language in P and E a nice code. By Theorem 31, (E, L) has proofs of polynomial size that can be computed in polynomial time and verified by a coRPOLYLOG machine V_L .

Given an input $x \in E$ of length n and an integer k , we look for k -compressible proofs of membership of x in L , where a k -compressible proof is the truth table of a Boolean circuit of size at most $(\log(n))^k$. If $x \notin L$, no such proofs exist, i.e., any such purported proof is rejected by V_L with high probability. If $x \in L$, a k -compressible proof may or may not exist. And if there are inputs in $L \cap E$ without k -compressible proofs, such inputs may or may not be easy to generate. We consider two cases:

1. For each language $L \in \text{P}$, there exists a positive integer k such that large inputs in $L \cap E$ without k -compressible proofs are hard to generate. More precisely, no polynomial-time weak refuter can produce an input in $L \cap E$ without a k -compressible proof for infinitely many input lengths. In this case, we will argue that $(E, L) \in \text{quasi}_{\text{P}}\text{-MAPOLYLOG}$ for all $L \in \text{P}$.
2. There exists a language $L \in \text{P}$ such that for each positive integer k there exists a refuter N_k operating in polynomial time which for infinitely many n produces at least one input of length n in $L \cap E$ without a k -compressible proof. In this case, we will argue that $\text{EXP} \not\subseteq \text{SIZE}(\text{poly})$.

Case 1. Consider the following machine M_k that tries to decide L on input x : M_k first guesses a circuit C_k of size less than $(\log(n))^k$, which takes inputs of length $O(\log(n))$. The circuit C_k is to be interpreted as a “ k -compressed” representation of a purported proof for x ; i.e., the output of the circuit on input i is the i th bit of the proof. Since the size of proofs can be polynomially bounded, the input to C_k need only be of length $O(\log(n))$. M_k then simulates V_L as follows: When V_L accesses a bit

of the input x , M_k accesses the same bit; when V_L accesses a bit of the (purported) proof, M_k gives the address of the bit to C_k as input, runs C_k , and continues its simulation of V_L , assuming the value of the bit to be the answer given by C_k .

We now argue that under the Case 1 hypothesis, when we view M_k as a Merlin-Arthur simulation of (E, L) , no polynomial-time weak refuter succeeds against M_k . Since machine M_k runs in polylogarithmic time,³ we can conclude that $(E, L) \in \text{quasi}_P\text{-MAPOLYLOG}$.

Consider an arbitrary polynomial-time weak refuter N . We need to show that, for large enough n , M_k does not make a simulation error on any $x \in N(0^n)$ for the simulation of (E, L) in MAPOLYLOG. By assumption, for large enough n , all $x \in N(0^n)$ which belong to $L \cap E$ have k -compressible proofs, and M_k accepts with probability 1 for such inputs by guessing the correct circuit corresponding to the proof and simulating the verifier V_L . Hence, for such inputs x , M_k conforms to MAPOLYLOG on x and agrees with L on x , which implies that it does not make a simulation error. For $x \in N(0^n)$ that belong to $E \setminus L$, any circuit guessed by M_k corresponds to an invalid proof, which is rejected with high probability. Hence again, M_k conforms to MAPOLYLOG on x and agrees with L on x , which means that M_k does not make a simulation error. Thus, for large enough n , M_k does not make a simulation error on any $x \in N(0^n) \cap E$. In other words, the simulation M_k of (E, L) in MAPOLYLOG fools N .

Case 2. For each positive integer k , we will define a polynomial-time procedure Q_k which, given 0^n as input, outputs in time $\text{poly}(n)$ the truth table of a Boolean function on inputs of length $O(\log(n))$, which does not have circuits of size $(\log(n))^{k-1}$, for infinitely many n . Suppose for a moment that we have such a procedure Q_k . We then define a language $L_k \in \mathbf{E}$ such that, for infinitely many n , L_k does not have circuits of size $O(n^{k-2})$ on inputs of length n . Let the size of the truth table output by Q_k on input 0^n be n^c for some constant $c > 1$. We define the truth table of L_k on input length n to be the concatenation of the outputs of Q_k on inputs $0^i, i = 2^{n/2^c}, \dots, 2^{(n+1)/2^c}$, padded with 0's to size 2^n . Then $L_k \in \mathbf{E}$, and L_k infinitely often does not have circuits of size $(n/3c)^{k-1}$, and hence infinitely often does not have circuits of size $O(n^{k-2})$. Note that the language L_k depends on Q_k . We obtain a fixed language L that does not have circuits of size $O(n^{k-2})$ for any k by considering any linear-time complete language for \mathbf{E} : If such a language L had circuits of size n^{k-2} , then L_k would have circuits of size $O(n^{k-2})$, which is a contradiction.

We finish the proof by constructing the procedure Q_k with the required properties. Let (E, L) have proofs of length $p(n)$, and let $f : \{0, 1\}^* \rightarrow \{0, 1\}^*$ be a polynomial-time computable function that for each $x \in L \cap E$ outputs a proof for x of length $p(n)$. Let N_k be the refuter from the Case 2 hypothesis, which is supposed to output for infinitely many n a list of inputs of length n , at least one of which is in $L \cap E$ and has no k -compressible proof. Let $q_k(n)$ denote the running time of N_k . On input 0^n , Q_k first simulates N_k on 0^n to get a list of strings x_1, x_2, \dots , where each string is of length n and the size of the list is less than $q_k(n)$. Q_k concatenates the strings $f(x_1), f(x_2), \dots$ into a single string that is of size at most $p(n)q_k(n)$ and outputs this string. Since f is polynomial-time computable, Q_k works in polynomial time. It remains to show that the string output by Q_k is the truth table of a function that does not have circuits of size $(\log(n))^{k-1}$ for infinitely many n . Suppose this were false. Since each $f(x_i)$ is an easily identifiable part of the truth table output by Q_k on input 0^n , a circuit for $f(x_i)$ (interpreted as a truth table) of size $(\log(n))^k$ can be

³Note that a polylogarithmic-time Turing machine can determine the length of its input and verify that the length of the guessed string C_k is correct.

derived from the circuit of size $(\log(n))^{k-1}$ for the truth table output by Q_k . Thus, each output x_i of N_k that lies in $L \cap E$ has a k -compressible proof. This contradicts the Case 2 hypothesis. \square

3.1.2. Comparison of main result with Nisan's result. We now compare the consequent of Theorem 1 and the statement $\text{EXP} = \text{MA}$, which was the strongest previously known consequence, shown by Nisan, of the hypothesis that EXP has polynomial-size circuits. Standard translation techniques show that the statement $\text{EXP} = \text{MA}$ is equivalent to the existence of simulations of P in MAPOLYLOG that fool the tally refuter. We include the proof for completeness.

LEMMA 33. $\text{EXP} = \text{MA}$ iff $\text{P} \subseteq \text{pseudo}_{\text{Tally}}\text{-MAPOLYLOG}$.

Proof. By padding, $\text{EXP} = \text{MA}$ iff $\text{E} \subseteq \text{MA}$. We show that $\text{E} \subseteq \text{MA}$ iff $\text{P} \subseteq \text{pseudo}_{\text{Tally}}\text{-MAPOLYLOG}$.

For the forward direction, let L be a language in P . Let $L' = \{n \mid 0^n \in L\}$. L' is in E , and by hypothesis, there is a Merlin-Arthur machine M' running in polynomial time accepting L' . Now we define a Merlin-Arthur machine M running in polylogarithmic time as follows: M finds the length n of its input (which can be done in deterministic polylogarithmic time) and runs M' on input n . By construction, M accepts a string of the form 0^n iff such a string is in L . Thus the simulation of L by M fools the tally refuter.

For the converse, let $L \in \text{E}$. Consider the language $L' = \{0^n \mid n \in L\}$. L' is in P , and hence, by the hypothesis, there is a machine M' which runs in Merlin-Arthur polylogarithmic time on all inputs of the form 0^n and accepts exactly those inputs of the form 0^n which are in L' . We define a Merlin-Arthur machine M , which, given n , runs M' on the input 0^n . Whenever M' requests the i th bit of its input, M checks whether $i \leq n$. If so, M continues the simulation of M' , assuming that the input bit is 0; otherwise it continues the simulation of M' by notifying M' that the requested input bit does not exist. M is a Merlin-Arthur machine running in polynomial time accepting L , and thus $L \in \text{MA}$. \square

We next argue that the existence of a polynomial-time computable linear code E such that $(E, L) \in \text{quasi}_{\text{P}}\text{-MAPOLYLOG}$ for each $L \in \text{P}$ implies that $\text{EXP} = \text{MA}$, or equivalently that $\text{P} \subseteq \text{pseudo}_{\text{Tally}}\text{-MAPOLYLOG}$. The reason is that the tally refuter essentially is a special case of a weak refuter restricted to such a code E .

THEOREM 34. *If there is a polynomial-time computable linear code E such that for each language $L \in \text{P}$, $(E, L) \in \text{quasi}_{\text{P}}\text{-MAPOLYLOG}$, then $\text{EXP} = \text{MA}$.*

Proof. The proof is again via a translation argument. Let E be a polynomial-time computable linear nice code mapping $\{0, 1\}^n$ to $\{0, 1\}^{p(n)}$ for some polynomial $p(n)$. By linearity of E , $0^{p(n)}$ is a codeword of E for each integer n .

We show $\text{E} \subseteq \text{MA}$ under the hypothesis, which implies $\text{EXP} = \text{MA}$ by padding. Let $L \in \text{E}$. Consider the language $L' = \{0^{p(n)} \mid n \in L\}$. Since $p(n)$ is polynomial-time computable (by the polynomial-time computability of E), $L' \in \text{P}$. Hence there is a machine M' which for each weak refuter N runs in Merlin-Arthur polylogarithmic time on strings produced by N , which are codewords of E , and agrees with L' on such strings. In particular, M' is a successful simulation of (E, L') in MAPOLYLOG against the tally refuter. As in the proof of Lemma 33, we define a Merlin-Arthur polynomial-time machine M accepting L . Hence, $L \in \text{MA}$. \square

Since there exists a polynomial-time computable linear nice code (Theorem 32), Theorem 34 shows that the consequent of Theorem 1 implies that $\text{EXP} = \text{MA}$. Thus, Theorem 1 is at least as strong as Nisan's result. Since weak refuters restricted to a polynomial-time computable linear nice code E are more powerful than the tally

refuter (which is the weakest possible refuter), Theorem 1 seems stronger than Nisan's result.

3.1.3. Nonuniform simulations against weak refuters. We should point out that very efficient simulations against polynomial-time weak refuters restricted to a nice code E exist unconditionally in the *nonuniform* setting. The following result states that for each language L and each nice code E there is a simulation of (E, L) by decision trees of depth $O(\text{polylog}(n))$ that fools every polynomial-time weak refuter. Recall that *qpoly* refers to advice of quasi-polynomial length, i.e., length $2^{O(\text{polylog}(n))}$.

THEOREM 35. *For each language L and nice code E , $(E, L) \in \text{quasiP-DPOLYLOG/qpoly}$.*

Proof. The idea of the proof is that the only codewords of E that a weak refuter can produce are of low time-bounded Kolmogorov complexity and also, because E is an error-correcting code, elements from E can be distinguished from each other by looking at only a few selected bit positions. Since there are only few strings of low time-bounded Kolmogorov complexity, membership information about them can be determined from a table which is given as a quasi-polynomial length advice string.

We briefly describe the notion of time-bounded Kolmogorov complexity that we need. Fix a standard deterministic universal Turing machine U , which takes an argument p , runs the program p on the empty string, and outputs the result. We consider Levin's measure $Kt(x) = \min\{|p| + \log(t) \mid p \text{ outputs } x \text{ in } t \text{ steps}\}$. We need two observations:

1. Each string output by a polynomial-time weak refuter has $O(\log(n)) Kt$ -complexity. This is because the position of the string in the list of strings output by the refuter serves as a description of the string. Since the refuter can be described by $O(1)$ bits and the position by $O(\log(n))$ bits, this description is $O(\log(n))$ bits long.
2. There are at most 2^l strings x with $Kt(x) < l$ for any l .

The second item holds irrespective of which notion of Kolmogorov complexity we use, but Levin's measure is convenient if we require the first item to hold as well.

Let L be any language and E be a nice code. Let k be a positive constant such that any two distinct codewords of length m in E have relative distance at least $1/(\log(m))^k$. Let $\epsilon < 1$ be a positive constant. For each n , we define $EZ(n) = \{x \in E \mid |x| = n, Kt(x) \leq (\log(n))^{1+\epsilon}\}$. Given any polynomial-time weak refuter, each codeword of E of length n output by the weak refuter is in $EZ(n)$ for all but finitely many n , by observation 1 above. Also, $|EZ(n)| \leq 2^{(\log(n))^{1+\epsilon}}$, by observation 2.

For any n , we consider the following experiment: pick positions $i_1, \dots, i_{(\log(n))^{k+2}}$ uniformly and independently at random between 1 and n . We claim that with probability at least $1/2$, any distinct u and v in $EZ(n)$ differ in at least one of these positions. To see this, let u and v be a specific pair of distinct codewords in E . E has relative distance bounded below by $1/(\log(n))^k$; hence with probability at least $1 - (1 - 1/(\log(n))^k)^{(\log(n))^{k+2}}$, u and v differ in at least one of the positions $i_1, \dots, i_{(\log(n))^{k+2}}$. By a union bound, since $|EZ(n)| \leq 2^{(\log(n))^{1+\epsilon}}$, the probability that there are distinct u and v in $EZ(n)$ which do not differ in any of the positions is at most $1/2$. Thus the claim is proved.

The probabilistic argument above shows that there is some sequence of positions $i_1, \dots, i_{(\log(n))^{k+2}}$ between 1 and n such that every two distinct codewords in $EZ(n)$ differ in at least one of these positions. Fix such a sequence. For each n , we code this sequence into the advice string for a DPOLYLOG machine simulating (E, L) . We also code into the advice string a table which contains, for each possible assignment

of (0,1)-values to these positions, the value 1 if there is $u \in E \cap L$ of length n such that u is consistent with that assignment of (0,1)-values to positions, and the value 0 otherwise. The length of the advice string is $O(2^{(\log(n))^{k+2}})$.

We now describe how a DPOLYLOG machine M simulating (E, L) behaves, given an advice string as above. M , on input x of length n , first determines $i_1, \dots, i_{(\log(n))^{k+2}}$ from its advice string. It then reads these positions $x_{i_1}, \dots, x_{i_{(\log(n))^{k+2}}}$ and looks up the corresponding entry in the table coded into the advice string. If the entry is 1, it accepts; otherwise it rejects.

We need to show that this simulation succeeds against all polynomial-time weak refuters. Let N be a polynomial-time weak refuter. Without loss of generality, we can assume that N produces only codewords of E , since E is a nice code, and N cannot fool the simulation by producing a noncodeword. Let x be a string of length n produced by N . $x \in EZ(n)$, and by the choice of positions $i_j, 1 \leq j \leq (\log(n))^{k+2}$, the bits of x in these positions determine x uniquely among all strings in $EZ(n)$. Thus the entry in the table coded in the advice string of M corresponding to the assignment $x_{i_1}, \dots, x_{i_{(\log(n))^{k+2}}}$ of (0,1)-values to bit positions tells correctly whether $x \in L$ or not, and hence the simulation succeeds on input x . This argument works for any large enough n , and hence the theorem follows. \square

Arguably, Theorem 35 points to a weakness in the kind of simulations of promise problems we consider. Note, though, that nonuniformity of the simulating class plays a crucial role. In fact, a diagonalization argument as in the proof of the time hierarchy theorem yields a language $L \in \mathsf{P}$ such that $(E, L) \notin \text{quasi-P-DPOLYLOG}$ for each nice code E . Thus, we view Theorem 35 as an indication of the crucial role nonuniformity plays in Theorem 1 and in Nisan's argument.

Nevertheless, it would be more pleasing if we could lift the restriction to a nice code E in Theorem 1. However, we can prove unconditionally that $\mathsf{P} \not\subseteq \text{quasi-P-MAPOLYLOG}$.

THEOREM 36. $\mathsf{P} \not\subseteq [\text{io} - \text{quasi-P}]\text{-MAPOLYLOG}$.

Proof. Consider the language $L = \{x \mid x \text{ has an even number of 1's}\}$. We show that $L \notin [\text{io} - \text{quasi-P}]\text{-MAPOLYLOG}$. The basic idea of the proof is that the behavior of a polylogarithmic-time Merlin-Arthur machine cannot change very much if one bit of the input is changed. On the other hand, changing a bit of the input makes the difference between membership in L and nonmembership in L . Thus, given a set comprising an input $x \notin L$ and all inputs at Hamming distance 1 from x (all of which are in L), every polylogarithmic-time Merlin-Arthur machine must fail to behave correctly on at least one input in the set. We observe that there is such a set that can be generated in polynomial time.

For any purported simulation of L in MAPOLYLOG, we use the same weak refuter, which outputs the string of all 0's and all strings with exactly one 1. Clearly, the weak refuter can do this in polynomial time. We show that for every machine M , either there is an input of length n produced by the weak refuter such that M does not conform to (bounded two-sided error) Merlin-Arthur polylogarithmic time on the input, or there is an input of length n such that M does not agree with L on the input, for almost all input lengths n . If M rejects the all 0's input or if the computation of M does not conform to Merlin-Arthur polylog time on the all 0's input, then we are done. Otherwise, fix an accepting nondeterministic path p of M on the all 0's input. At most a polylogarithmic number of different input bits can be queried by this path—assume without loss of generality that each of these is one of $x_1, x_2, \dots, x_{\text{polylog}(n)}$. Now consider the associated subtree of randomized nodes,

namely the subtree of configurations accessible by M , given that it has made the nondeterministic choices corresponding to the path p . At least a fraction $2/3$ of the paths in this subtree end in an accepting node. Since only a polylogarithmic number of input bits can be queried on any one path, there is some $i, i > \text{polylog}(n)$, for which x_i is queried on less than a fraction $\text{polylog}(n)/n$ of the paths. Now, if x_i is flipped, p is still a valid path for M to take, since x_i is not queried on this path. Moreover, in the subtree of randomized nodes associated with p , at least a $2/3\text{-polylog}(n)/n$ fraction of the paths still end in accepting nodes since they do not query x_i , and all the other bits are unchanged. Thus either the computation of M does not conform to Merlin-Arthur polylog time on the input with all 0's except a 1 in the i th position, or at least $2/3$ of the paths of this subtree end in accepting nodes, which means that M accepts the input. In this case, M does not agree with L on the string with all 0's except a 1 in the i th position. \square

The result above holds even if we consider nonuniform MAPOLYLOG, i.e., the machine is also allowed a quasi-polynomial amount of advice (to which random access is available).

3.1.4. Simulations without restriction to a nice code. If we allow a slightly larger simulating class than MAPOLYLOG in the statement of Theorem 1, we *can* lift the restriction to a nice code, as follows.

THEOREM 37. *If $\text{EXP} \subseteq \text{SIZE}(\text{poly}(n))$, then $\text{P} \subseteq \text{quasi}_{\text{P}}\text{-}\Sigma_2\text{-POLYLOG}$.*

Proof sketch. The proof follows the same techniques as the proof of Theorem 1, except that instead of Theorem 31, we use the fact that P has proofs of polynomial size verifiable in coNPOLYLOG . This holds because the tableau of the computation of a deterministic Turing machine on an input is a valid proof if the input is in the language, and local consistency of the tableau can be checked in coNPOLYLOG . Since, for each $L \in \text{P}$, proofs verifiable in coNPOLYLOG exist for *all* inputs in L , not just inputs that are codewords of a nice code, we do not require the restriction to a nice code in the statement of the theorem. \square

Note that while $\text{P} \not\subseteq \text{quasi}_{\text{P}}\text{-MAPOLYLOG}$, as shown in Theorem 36, showing nonexistence of simulations in a slightly larger class or against somewhat weaker adversaries would imply that EXP does not have polynomial-size circuits.

Analogously to Theorem 35, we can show that the consequent of Theorem 37 holds unconditionally if the simulating class is nonuniform $\Sigma_2\text{-POLYLOG}$ rather than uniform $\Sigma_2\text{-POLYLOG}$.

THEOREM 38. *$\text{P} \subseteq \text{quasi}_{\text{P}}\text{-}\Sigma_2\text{-POLYLOG}/\text{qpoly}$.*

Proof sketch. As in the proof of Theorem 35, we use the fact that polynomial-time weak refuters can only produce strings of low time-bounded Kolmogorov complexity. Let L be a language in P . We define a nonuniform $\Sigma_2\text{-POLYLOG}$ machine which simulates L . The advice string for M at input length n is an enumeration of all $x \in L$ of length n with $Kt(x) \leq (\log(n))^{1+\epsilon}$, for some constant ϵ . The length of the advice string is thus $O(2^{(\log(n))^{1+\epsilon}})$. The machine M operates as follows: given an input x of length n , it tests whether x is the same as some string for which membership in L is coded into the advice string. This is done in $\Sigma_2\text{-POLYLOG}$ by existentially guessing a segment of the advice string and universally verifying that the segment matches x bit-by-bit.

Note that all strings of length n produced by a weak refuter have time-bounded Kolmogorov complexity $O(\log(n))$ and hence are accounted for by the advice string for large enough n . Thus the simulation succeeds against all polynomial-time weak refuters. \square

3.2. Variants. We can modify the parameters in Theorem 1 in several ways. First, under the same hypothesis that $\text{EXP} \subseteq \text{SIZE}(\text{poly}(n))$, we can trade off the simulation time and the running time of the adversary.

THEOREM 39. *If $\text{EXP} \subseteq \text{SIZE}(\text{poly}(n))$, then for each $L \in \text{P}$, each nice code E and each $\epsilon > 0$, $(E, L) \in \text{quasi}_{\text{SUBEXP}}\text{-MATIME}(n^\epsilon)$.*

Proof. The proof is analogous to the proof of Theorem 1. For each ϵ and each language $L \in \text{P}$, we define a simulation of (E, L) in $\text{MATIME}(n^\epsilon)$ as before. If there is a language L and an $\epsilon > 0$ such that there is some subexponential-time weak refuter not fooled almost everywhere by the simulation, the concatenation of the proofs of the strings output by the refuter is the truth table of a function that does not have polynomial-size circuits. \square

Similarly, we can prove the following result.

THEOREM 40. *If $\text{EXP} \subseteq \text{SIZE}(\text{poly}(n))$, then for each $\epsilon > 0$, $\text{P} \subseteq \text{quasi}_{\text{SUBEXP}}\text{-}\Sigma_2\text{-TIME}(n^\epsilon)$.*

We can also use weaker hypotheses than $\text{EXP} \subseteq \text{SIZE}(\text{poly}(n))$, and obtain weaker simulations.

THEOREM 41. *If $\text{E} \subseteq \text{SIZE}(2^{\epsilon n})$ for each constant $\epsilon > 0$, then for each time bound $t = n^{\Omega(1)}$, each language $L \in \text{DTIME}(t)$, each nice code E decidable in time $O(t \text{ polylog}(t))$, and each $\delta > 0$, $(E, L) \in \text{quasi}_{\text{P}}\text{-MATIME}(t^\delta)$.*

Note that by Theorem 32, quasi-linear time decidable nice codes exist.

If the assumptions in Theorems 1 and 41 hold only for infinitely many input lengths (instead of almost all), the simulation is correct on infinitely many input lengths.

It is not clear whether the consequent of Theorem 40, that for all $\epsilon > 0$, $\text{P} \subseteq \text{quasi}_{\text{SUBEXP}}\text{-}\Sigma_2\text{-TIME}(n^\epsilon)$, is unlikely to hold. We show below that the negation of this statement holds under a reasonable circuit lower bound assumption. Thus, we obtain a partial converse to Theorem 40. Interestingly, the nonuniform methods used to show circuit lower bounds for AC^0 circuits [Ajt83, FSS84, Hås86] come in useful here.

LEMMA 42. *There is a constant $\epsilon > 0$ such that $\text{P} \not\subseteq [\text{io-corr}_{2/3}]\text{-}\Sigma_2\text{-TIME}(n^\epsilon)$.*

This result immediately follows from the inapproximability of PARITY by subexponential-size constant-depth circuits [Hås86]. Note that $\Sigma_2\text{-TIME}(n^\epsilon)$ can be simulated by depth- d circuits of size 2^{n^ϵ} for some fixed constant d .

The average case hardness result of Lemma 42 has the following consequence for simulations against probabilistic weak refuters.

THEOREM 43. *There is a constant $\epsilon > 0$ such that $\text{P} \not\subseteq [\text{io-quasi}]_{\text{BPP}}\text{-}\Sigma_2\text{-TIME}(n^\epsilon)$.*

Proof. Let L be a language in P such that there is an $\epsilon > 0$ for which $L \notin [\text{io-corr}]_{2/3}\text{-}\Sigma_2\text{-TIME}(n^\epsilon)$. By Lemma 42, such a language L exists. Given a purported $\Sigma_2\text{-TIME}(n^\epsilon)$ simulation L' of L , the strategy of the probabilistic weak refuter is simple and, in fact, independent of L' : it just picks $c = O(1)$ strings of input length at random and outputs them. There is a large enough c so that, for at least $2/3$ of the random choices of the probabilistic adversary, at least one of the strings it outputs on this random choice is in $L \triangle L'$. \square

Our strategy now is to show that under appropriate circuit lower bound assumptions, the probabilistic weak refuter can be derandomized, i.e., replaced by a deterministic weak refuter. The proof will use the following derandomization lemma, which shows how probabilistic computations satisfying certain conditions can be derandomized under assumptions related to the complexity of testing for the conditions.

LEMMA 44 (see [KvM02]). *Let $D(x, r) : \{0, 1\}^n \times \{0, 1\}^{p(n)} \rightarrow \{0, 1\}^*$ be a polynomial-time computable function and $C(y, r)$ a predicate such that $C(D(x, r), r) = 1$ for at least $2/3$ of the random strings r . Let C be computable in $\text{SIZE}^A(\text{poly}(n))$, for some oracle A . If $\text{E} \not\subseteq \text{SIZE}^A(\text{poly}(n))$, then there is a set S of strings such that $S \cap \{0, 1\}^{p(n)}$ is enumerable in time 2^{n^ϵ} for any $\epsilon > 0$, and for infinitely many n , $C(D(x, r'), r') = 1$ for at least one of the strings r' of length $p(n)$ in S .*

THEOREM 45. *If $\text{E} \not\subseteq \text{SIZE}^{\Sigma_2\text{-POLY}}(\text{poly}(n))$, then there is a constant $\epsilon > 0$ such that $\text{P} \not\subseteq \text{quasi}_{\text{SUBEXP-}\Sigma_2\text{-TIME}}(n^\epsilon)$.*

Proof. From the proof of Theorem 43, we know that there is a language $L \in \text{P}$, a probabilistic weak refuter M , and a constant $\epsilon > 0$ such that for any language $L' \in \Sigma_2\text{-TIME}(n^\epsilon)$, M weakly distinguishes L from L' . We would like to transform M to a deterministic weak refuter M' for which the same property holds, under an appropriate assumption. Intuitively, this is a “derandomization” because we are trying to eliminate the randomness of the machine M . However, the derandomization needs to preserve the distinguishing properties of M . We apply Lemma 44 in order to do this.

In our application of Lemma 44, $p(n)$ is the number of random bits used by the probabilistic weak refuter, the function $D(0^n, r)$ specifies which string of length n is output by the refuter on a given sequence of random bits r , and the predicate C tests whether a given string output by the refuter is in the symmetric difference of L and L' . Since both L and L' are in $\Sigma_2\text{-POLY}$, C is computable in $\text{SIZE}^{\Sigma_2\text{-POLY}}(\text{poly}(n))$. Thus, by the theorem, if $\text{E} \not\subseteq \text{SIZE}^{\Sigma_2\text{-POLY}}(\text{poly}(n))$, there is a set S of strings such that $S \cap \{0, 1\}^{p(n)}$ is enumerable in time 2^{n^ϵ} for each $\epsilon > 0$ and $D(0^n, r')$ is in the symmetric difference of L and L' for at least one of the strings $r' \in S \cap \{0, 1\}^{p(n)}$. We define the deterministic weak refuter M' to first generate $S \cap \{0, 1\}^{p(n)}$ on input 0^n and output the set of strings $\{D(0^n, r') \mid r' \in S \cap \{0, 1\}^{p(n)}\}$. It follows from the properties of M and Theorem 44 that M' weakly distinguishes L from L' infinitely often. Also, if $S \cap \{0, 1\}^{p(n)}$ is enumerable in time 2^{n^ϵ} for any $\epsilon > 0$, then by using an appropriate enumerator for $S \cap \{0, 1\}^{p(n)}$, M' can be defined to operate in time 2^{n^ϵ} for any $\epsilon > 0$, and hence the theorem follows. \square

Theorem 45 can be considered a partial converse to Theorem 40.

3.3. Consequences. In a straightforward way, the fact that circuit lower bounds for EXP imply derandomization translates to tradeoffs between simulations of P in low-level complexity classes that fool uniform adversaries and deterministic simulations of probabilistic classes.

THEOREM 46 (see [BFNW93]). *If $\text{EXP} \not\subseteq \text{SIZE}(\text{poly})$, then $\text{BPP} \subseteq i.o.\text{SUBEXP}$.*

Corollary 2 follows from Theorems 1 and 46.

THEOREM 47 (see [IW97]). *If $\text{E} \not\subseteq \text{SIZE}(2^{\epsilon n})$, for some $\epsilon > 0$, then $\text{BPP} \subseteq i.o.\text{P}$.*

COROLLARY 48. *For each $\epsilon > 0$ and time bound $t = n^{\Omega(1)}$, for each language $L \in \text{DTIME}(t)$ and each nice code E decidable in time $O(t \text{ polylog}(t))$, $(E, L) \in \text{quasi}_{\text{P-MATIME}}(t^\epsilon)$, or $\text{BPP} \subseteq i.o.\text{P}$.*

Because $\text{MATIME}(t)$ is trivially contained in $\text{DSPACE}(t)$, our results also generalize the space-randomness tradeoffs first shown by Sipser [Sip88] and later extended by Nisan and Wigderson [NW94] and Lu [Lu01]. Sipser showed that, conditioned on the explicit constructibility of a certain kind of disperser (later shown to hold in [SSZ98]), if there does not exist an $\epsilon < 1$ such that for all polynomially bounded t , $\text{DTIME}(t) \subseteq [i.o - \text{pseudo}_{\text{Tally}}]\text{-DSPACE}(t^\epsilon)$, then $\text{RP} = \text{P}$. Nisan and Wigderson [NW94] extended this tradeoff to other ranges of parameters, and Lu [Lu01] obtained derandomizations of AM under the same assumptions. We consider simu-

lations against polynomial-time adversaries producing inputs in a nice code E , while the aforementioned papers considered only tally adversaries.⁴

3.4. Extensions. We can prove an analogue of Theorem 1 for space, as follows.

THEOREM 49. *If $\text{PSPACE} \subseteq \text{SIZE}(\text{poly})$, then for each language $L \in \text{LOGSPACE}$ and each logspace-decidable nice code E , $(E, L) \subseteq \text{quasi}_{\text{LOGSPACE}}\text{-MAPOLYLOG}$.*

Proof sketch. The proof follows along the same lines as the proof of Theorem 49, except that we use the analogue of Theorem 31 for LOGSPACE, which states that for each language $L \in \text{LOGSPACE}$ and each logspace-decidable nice code E , (E, L) has proofs of size $\text{poly}(n)$ verifiable in coRPOLYLOG , and given $x \in L \cap E$, a proof that $x \in L$ can be computed in logarithmic space. \square

4. Simulations of nondeterministic time against uniform adversaries.

Using the ideas of section 3, we can show tradeoffs between simulations of nondeterministic time against uniform adversaries and simulations of probabilistic time in nondeterministic time against nonuniform adversaries. The translation is straightforward except for two issues.

The first issue is that the hypothesis $\text{NEXP} \not\subseteq \text{SIZE}(\text{poly})$ is not known to yield subexponential nondeterministic simulations of probabilistic computations. What does suffice for the latter is a way to generate in nondeterministic exponential time the truth table of a function that requires large circuits. In particular, we have the following result.

PROPOSITION 50. *If for each k there is an NP machine M_k that accepts all inputs of the form 0^{2^n} , and outputs the truth table of a function on n inputs that does not have circuits of size n^k on each accepting path, then $\text{MA} \subseteq \text{NSUBEXP}$.*

It is open whether $\text{NEXP} \not\subseteq \text{SIZE}(\text{poly})$ implies that we can generate in nondeterministic polynomial time the truth table of a function that requires large circuits.⁵ However, the proof strategy of Theorem 1 actually yields the latter, so we can obtain the derandomization results directly from it.

The other issue is that we can no longer assume that the ability to generate a list of inputs, such that at least one of them is in the language and has no easy witnesses, implies the ability to generate the truth table of a hard function. The reason is that a nondeterministic computation cannot verify that it guessed witnesses for *all* inputs on the list that are in the language, and therefore may miss the one that has no easy witnesses. In order to remedy this problem, we weaken the notion of uniform simulation by requesting that the adversary outputs a single input. In our terminology, we switch from “quasi-” simulations to the weaker “pseudo-” simulations considered in other papers.

The following result is analogous to Corollary 2, except that the i.o. quantifier is switched: It appears in the simulation against refuters rather than in the derandomization.

THEOREM 51. *For each language $L \in \text{NP}$ and each nice code E , $(E, L) \in [\text{io} - \text{pseudo}]_{\text{NP}}\text{-MAPOLYLOG}$ or $\text{MA} \subseteq \text{NSUBEXP}$.*

Proof sketch. The simulation is the same as the one in Theorem 1, and the proof uses the holographic proofs of [BFLS91]. The condition we need for generation of proofs now is that a proof for x can be generated from an NP-witness in polynomial time. If the simulation doesn’t succeed against a restricted nondeterministic strong

⁴[NW94] and [Lu01] actually considered simulations of exponential time, but this is equivalent to simulations of polynomial time against tally adversaries.

⁵The converse is known to hold [IKW02].

refuter, the strong refuter can guess correctly a codeword of E on which the simulation fails, guess an accepting computation on that input (since the simulation fails on an input only if the input is in the NP language and in E but not in the simulating language), compute a proof from this computation, and output the proof. The failure of the simulation implies that every proof output on an accepting computation of the strong refuter is hard, and these hard proofs can be used to derandomize MA. \square

A tradeoff result involving simulations against unrestricted strong refuters can be shown in a similar way to Theorem 37, as follows.

THEOREM 52. $\text{NP} \subseteq [\text{io} - \text{pseudo}]_{\text{NP}-\Sigma_2}\text{-POLYLOG}$ or $\text{MA} \subseteq \text{NSUBEXP}$.

This is a strengthening of a theorem in [Kab01], where simulations of NP in deterministic time are considered.

Analogous to Corollary 48, we have the following.

THEOREM 53. For each $\epsilon > 0$, each language $L \in \text{NP}$, and each nice code E , $(E, L) \in [\text{io} - \text{pseudo}]_{\text{NP}-\text{MATIME}(n^\epsilon)}$ or $\text{MA} \subseteq \text{NP}$.

Note that we are not able to switch the quantifiers a.e. and i.o. as we were able to in the deterministic case of Corollaries 2 and 48. The reason is the same as that for why we can only deal with strong refuters and not with weak ones. However, as in [IKW02], the simulation of MA can use a small advice string to cue it to an input on which the refuter succeeds.

THEOREM 54. For each language $L \in \text{NP}$ and each nice code E , $(E, L) \in \text{pseudo}_{\text{NP}} - \text{MAPOLYLOG}$, or else for each $\epsilon > 0$, $\text{MA} \subseteq \text{i.o.}[\text{NTIME}(2^{n^\epsilon})/n^\epsilon]$.

Using upward translation, we obtain a “gap” theorem for NEXP, as follows.

COROLLARY 55 (see [IKW02]). $\text{NEXP} = \text{MA}$, or else for each $\epsilon > 0$, $\text{MA} \subseteq \text{i.o.}[\text{NTIME}(2^{n^\epsilon})/n^\epsilon]$.

By standard diagonalization techniques, the clauses in Corollary 55 are exclusive.

Under a slightly stronger hypothesis than in Theorem 51, we can derandomize the class AM. First, we need the following result from [KvM02] showing that certain circuit lower bounds imply a derandomization of AM.

THEOREM 56. If $\text{EXP} \not\subseteq \text{i.o.}\text{SIZE}^{\text{SAT}}(\text{poly}(n))$, then $\text{AM} \subseteq \text{NSUBEXP}$.

We obtain our derandomization result for AM using a variation on the easy witness method where, for $L \in \text{NP}$, an “easy” proof that $x \in L$ is a proof that has small SAT-oracle circuits, when the proof is considered as the truth table of a function.

THEOREM 57. $\text{NP} \subseteq [\text{io} - \text{pseudo}]_{\text{NP}-\Sigma_3}\text{-POLYLOG}$ or $\text{AM} \subseteq \text{NSUBEXP}$.

Proof sketch. We make a modification to the simulation of Theorem 51. We will use proofs verified by “local checkability,” as in the proof of Theorem 37. Given a language $L \in \text{NP}$, the simulating machine guesses a SAT-oracle circuit of small size, and then verifies that this circuit is the truth table of a valid proof by using the local checkability algorithm and querying the circuit when it needs a bit of the proof. The problem is that the machine also needs to be able to answer SAT queries in order to evaluate the circuit. This can be done using one additional alternation, by guessing the queries and answers to queries at the beginning of a computation path and verifying them on a parallel computation path. Since SAT is in nondeterministic quasi-linear time and the queries are of size $O(\text{polylog}(n))$, the overhead incurred is minimal. \square

5. Quantitative simulations. In this section, we consider a quantitative version of simulation, where a simulation is guaranteed to work on a certain number of inputs of any given input length. The simulations in previous sections correspond to the following situation: A simulation may fail on many inputs, but it is infeasible to find a place where this happens. The situation here is different: A simulation

can fail on only a limited fraction of the inputs, but it may be easy to find an input on which the simulation makes a mistake. By using the easy witness method and coupling simulations together on an input-by-input basis rather than an input-length-by-input-length basis, it is possible to prove nontrivial tradeoffs for certain ranges of the parameters. More precisely, if an input x in a language $L \in \mathsf{P}$ does not have a compressible proof, we can use its proof as the truth table of a hard function for the pseudorandom generator of Theorem 24 to derandomize the computation of a BPP machine on the same input x .

Our results are based on the following lemma, which is a corollary to Theorem 31.

LEMMA 58. *For each language $L \in \mathsf{P}$ and each polynomial-time invertible nice code E , $(E, E(L))$ has proofs of size $\text{poly}(n)$ verifiable in coRPOLYLOG , where $E(L) = \{E(x) \mid x \in L\}$. Furthermore, given $x \in L$, a proof that $E(x) \in E(L)$ is computable in time $\text{poly}(n)$.*

THEOREM 59. *Fix a function $s : N \rightarrow [0, 1]$ such that $s(n)2^n$ is an integer for all $n \in N$. There exists $t = \text{poly}(n)$ such that $\mathsf{P} \subseteq \text{corr}_s\text{-MATIME}(t)$ or $\text{BPP} \subseteq [\text{io} - \text{corr}_{1-s}]\text{-P}$.*

Proof. If for each polynomial t there is a language $L \in \mathsf{P}$ such that $L \notin \text{corr}_s\text{-MATIME}(t)$, we will show how to simulate $\text{BPTIME}(t')$ in $[\text{io} - \text{corr}_{1-s}]\text{-P}$ for each polynomial-time bound t' . Given t' , let $t = \max(n^d, (t')^k)$, where n^d is the time required to compute $E(x)$, given x of length n , and k is a constant to be determined later. By assumption, there is a language $L \in \mathsf{P}$ such that $L \notin \text{corr}_s\text{-MATIME}(t)$.

Consider the following simulation for L : For each x , compute $E(x)$, guess a circuit of size \sqrt{t} , and check, using the probabilistic polylogarithmic-time verifier for $E(L)$, whether the truth table defined by the circuit is a proof that $E(x) \in E(L)$. By the properties of the proof system for $E(L)$, for any input x which is not in $E(L)$ or has proofs with circuits of size $\leq \sqrt{t}$, this simulation operates in Merlin-Arthur time t (since there are only polylogarithmically many queries made to the circuit and it takes time at most \sqrt{t} to answer a query) and does not make an error on x .

By hypothesis, the simulation makes an error on x for at least a $1 - s$ fraction of strings x of length n for infinitely many input lengths n . We show how to use the failure of the simulation to obtain a deterministic simulation of any language R accepted by a probabilistic polynomial-time machine M running in time t' . Given an input x , compute a proof $P(x)$ that x is in L (using Lemma 58). Use $P(x)$ as a hardness source in Theorem 24 to produce a polynomial-size list of pseudorandom strings. Run M on each of these strings and output the majority result as the answer.

We show that this works if we set k large enough, e.g., $k = 4c$, where c is the constant in the statement of Theorem 24. For each string x of length n , there are two possible cases. The first is that $P(x)$ has no circuits of size \sqrt{t} , in which case by our choice of k , it can be used as a hardness source for the pseudorandom generator of Theorem 24 to produce $(t')^2$ pseudorandom bits that fool circuits of size $(t')^2$, and can therefore be used for the derandomization of $\text{BPTIME}(t')$ algorithms (which can be simulated by circuits of size $(t')^2$). The second case is that $P(x)$ has circuits of size \sqrt{t} , which implies that the simulation of L in $\text{MATIME}(t)$ does not make an error on input x . Since there are infinitely many input lengths n such that the simulation of L in $\text{MATIME}(t)$ fails on at least a fraction $1 - s$ of inputs of length n , the simulation of any $\text{BPTIME}(t')$ language in P succeeds on at least a fraction $1 - s$ of inputs of these input lengths, proving the theorem. \square

Note that the i.o. and a.e. quantifiers can be interchanged in the preceding theo-

rem.

We would like to increase the advantage for the deterministic simulation of probabilistic time, given that the simulation of deterministic time does not succeed with a certain other advantage. We can show how to do this for certain values of the parameter s . Recall that we obtain our deterministic simulation of a BPP machine R on input x by running the pseudorandom generator of Theorem 24 using a proof of membership of x in a language $L \in \mathsf{P}$ as the truth table of a supposedly hard function f . Whenever x does not have a compressible proof, the function f is indeed hard, and so the pseudorandom generator succeeds in derandomizing R on input x . In order to succeed on more inputs than just those without compressible proofs, we will not just use the proof for x but also the proofs of a polynomial number of other inputs y of the same length as x . If at least one of them does not have a compressible proof, the concatenation of the polynomially many proofs will be incompressible enough and define a hard function f that we can successfully use in the pseudorandom generator of Theorem 24 to derandomize R on input x . We would like to pick the y 's such that, for many inputs x , they hit the inputs of length n without compressible proofs. In order to do so, we pick the y 's as the neighbors of x in an appropriate expander graph.

The formal proof will use the following two lemmas. The first one states the property that forms the basis for amplification using expanders, and the second one provides a family of expanders with the properties we need.

LEMMA 60 (see [KPS85]). *Let $G = (V, E)$ be a D -regular graph with $V = \{0, 1\}^n$ and second largest eigenvalue $\lambda < D^{9/10}$. Let $s < 1$ be a constant. Let $A \subseteq V$ be a set of size greater than $s2^n$, and let $N(A)$ denote the neighborhood of A , i.e., the set of vertices of G with at least one neighbor in A . Then $|N(A)| \geq (1 - D^{-1/10}/s)2^n$.*

LEMMA 61 (see [KPS85]). *There is an explicit family of graphs $\{G\}_n$ such that G_n has 2^n vertices and is $D = \text{poly}(n)$ -regular with second largest eigenvalue less than $D^{9/10}$. Given a vertex v of G_n , the neighbor set of G is computable in time $\text{poly}(n)$.*

THEOREM 62. *Let $s < 1$ and $c > 1$ be constants. There is a polynomial time bound t such that $\mathsf{P} \subseteq \text{corr}_{1-s}\text{-MATIME}(t)$ or $\mathsf{BPP} \subseteq [\text{io} - \text{corr}_{1-1/n^c}]\text{-P}$.*

Proof. We improve the simulation of $\text{BPTIME}(t')$ in the proof of Theorem 59 by using the amplification technique of [KPS85], while using the same simulation of P by $\text{MATIME}(t)$.

The idea is to interpret the set of strings of length n as the vertices of a D -regular expander G with second largest eigenvalue less than $D^{9/10}$, where D is polynomial in n . Let $\{G\}_n$ be an explicit family of graphs as in Lemma 61. Given an input string x , the set of neighbors y_i , $i = 1, \dots, D$, of x in G_n is determined in polynomial time. The concatenation of the strings $P(y_i)$ is used as hardness source for the deterministic simulation (rather than the string $P(x)$ as in the proof of Theorem 59).

Let t depend on t' in the same way as in the proof of Theorem 59. If $\mathsf{P} \not\subseteq \text{corr}_{1-s}\text{-MATIME}(t)$, there is a language L such that the simulation in the proof of Theorem 59 makes an error on at least an s fraction of the inputs for infinitely many input lengths n . Let us call a string y of length n “good” if $P(y)$ can be used as a hardness source in the deterministic simulation of a language $R \in \text{BPTIME}(t')$. It follows from our hypothesis on L that at least an s fraction of the strings of length n are good for infinitely many input lengths n . Fix an n for which at least an s fraction of the strings are good, and let A_n be the set of good strings. By Lemma 60, $|N(A_n)| \geq (1 - 1/n^c)2^n$, where c is a constant depending on D . Now note that the deterministic simulation of R succeeds on all strings in $N(A_n)$. Thus the simulation succeeds on at least an $1 - 1/n^c$ fraction of strings of length n for infinitely many n , proving the theorem.

□

Note that the quantifiers a.e. and i.o. in the statement of Theorem 62 can be interchanged.

The advantage of the simulation of BPP can be traded off against the time required for the simulation, but we do not pursue this direction here. Instead, we would like to increase the advantage of the simulation even further while maintaining a polynomial running time. We can do so, provided that we switch the a.e. and i.o. quantifiers. In that case, we are no longer restricted in our search for inputs without compressible proofs to y 's of the same length as the input x on which we want to simulate the BPP machine R , but we can use y 's of length polynomially smaller than $|x|$. That allows us to use dispersers instead of expanders, and obtain an advantage exponentially close to 1.

Let us first recall the notion of a disperser. An (N, M, T) -disperser is a bipartite multigraph $G = (V, W, E)$ with $|V| = N$ and $|W| = M$ having the property that any subset of V having at least T vertices has a neighbor set of size at least $2M/3$.

We will use the dispersers provided by the following lemma.

LEMMA 63 (see [SSZ98]). *For all $a, b, 1 \geq a > b \geq 0$, for any $T \geq 2^{\log(N)^a}$ and $M \leq 2^{\log(N)^b}$, there is an explicit (N, M, T) -disperser $G = (V, W, E)$ such that each vertex in V has degree polylogarithmic in N and, for any vertex in V , the neighbor set of the vertex can be computed in time polynomial in $\log(M + N)$ and the size of the neighbor set.*

Proof of Theorem 3. The idea of the proof is similar to the idea of the proof of Theorem 62, except that we use disperser-based amplification rather than expander-based amplification to achieve stronger parameters in this case. We simulate a BPP algorithm R on input x of length n using the fact that the simulation of some language $L \in \mathbf{P}$ by Merlin-Arthur machines operating within some fixed polynomial-time bound has failed for at least $1/3$ of the strings on every input length smaller than n , specifically for input length n^δ , $\delta < \epsilon$. By Lemma 63, there is an explicit $(2^n, 2^{n^\delta}, 2^{n^\epsilon})$ -disperser $G = (V, W, E)$ such that neighbor sets of vertices in V are efficiently computable. We can define a deterministic polynomial-time machine S which simulates R by interpreting its input x as a vertex in V , computing the neighbors $x_1, \dots, x_{\text{poly}(n)}$ of x in G , computing proofs $P(x_1), \dots, P(x_{\text{poly}(n)})$ for these strings, concatenating the proofs together, and using the resulting string as a hardness source for the derandomization of R . It follows from the properties of the disperser G that S works correctly on all but 2^{n^ϵ} inputs of length n , for each n . □

Theorem 3 is a uniform strengthening of the following result by Goldreich and Wigderson [GW02]. Recall that $\text{MATIME}(t) \subseteq \text{SIZE}^{\text{SAT}}(\text{poly}(t))$.

THEOREM 64 (see [GW02]). *There is a polynomial time bound t such that $\mathbf{P} \subseteq [\text{io} - \text{corr}_{2/3}] - \text{SIZE}^{\text{SAT}}(t)$ or, for each $\epsilon > 0$, $\mathbf{BPP} \subseteq \text{corr}_{1-2^{n^\epsilon}/2^n} - \mathbf{P}$.*

Finally, we show a consequence of the hypothesis that \mathbf{P} cannot be simulated by $\text{MATIME}(t)$, where the notion of simulation used is the conventional one. Before stating and proving our result, we state the almost-everywhere hierarchy theorem for deterministic time [GHS91], which is used in the proof.

LEMMA 65 (see [GHS91]). *Let $T(n)$ and $t(n)$ be constructible time bounds such that $t(n) \log(t(n)) = o(T(n))$. Then there is a language D in $\text{DTIME}(T(n))$ such that neither D nor \overline{D} has an infinite subset in $\text{DTIME}(t(n))$.*

THEOREM 66. *There is a polynomial time bound t such that $\mathbf{P} \subseteq \text{MATIME}(t)$ or $\text{DTIME}(T) \not\subseteq \text{ZPP}$ for any superpolynomial time bound T .*

Proof. We shall show that, under the assumption that there is no polynomial-time

bound t such that $\mathsf{P} \subseteq \mathsf{MATIME}(t)$, for every language R in ZPP , either R has an infinite subset in P or \overline{R} has an infinite subset in P . Given this, the consequence that $\mathsf{DTIME}(T) \not\subseteq \mathsf{ZPP}$ follows from Lemma 65.

We follow the proof of Theorem 59 for $s(n)2^n = 2^n - 1$. Let R be a language in ZPP accepted by a zero-error probabilistic Turing machine M running in time t' . Let L be a language in P that is not in $\mathsf{MATIME}(t)$ for t sufficiently larger than t' but still polynomial, as set in the proof of Theorem 59. Since $L \notin \mathsf{MATIME}(t)$, there is an infinite set X of strings x such that $P(x)$ is a good hardness source for the simulation of M on input x : Using $P(x)$ as a hardness source in Theorem 24 yields a polynomial length list of strings, at least one of which is a ZPP witness for M on input x . Since we can check the validity of ZPP witnesses for M on input x in polynomial time, we have an infinite set X in P for which we can decide the membership in R in polynomial time. If $R \cap X$ is infinite, it is an infinite subset of R in P ; otherwise, $\overline{R} \cap X$ is infinite and forms an infinite subset of \overline{R} in P . \square

6. Unconditional simulations. The trivial deterministic simulation of a time t probabilistic multitape Turing machine computation runs in time $\Omega(2^t)$. All previously published $o(2^t)$ deterministic simulations work only under unproven lower bound assumptions. This raises the question whether anything nontrivial can be shown unconditionally.

The oracle relative to which there is a constant c such that $\mathsf{DTIME}(2^{cn}) \subseteq \mathsf{BPTIME}(n)$ [Hel86, KV96] can be considered a negative indication in this connection. Nevertheless, we show that nontrivial deterministic simulations *do* exist for multitape Turing machines, as well as for restricted formulae and NC^1 circuits.

As explained in the introduction, our technique involves recycling random bits. Essentially, a large initial prefix of the random string used by the probabilistic algorithm can be interpreted as a randomness source for an extractor to produce additional almost-random bits and thus save on randomness later in the computation. We refer to section 1.3 for a more detailed intuitive account of our approach. We now proceed with the formal proof.

For the sake of economy of presentation, we use the single extractor family given by Theorem 26 in the proofs of our results for Turing machines and NC^1 circuits. For each of our results, there are earlier constructions that have the properties needed for the proof of that particular result, but the construction from Theorem 26 has all these properties combined.

THEOREM 67. *For any integers $k > 0$ and $\beta > 0$, there is a constant $\delta > 0$ such that for each language L accepted by a probabilistic k -tape Turing machine with alphabet size β and constant error bound $\epsilon < 1/2$ in time t , L is accepted by a probabilistic Turing machine with error bound ϵ that runs in time $\text{poly}(t)$ and uses $(1 - \delta)t$ random bits.*

Proof. Let M be a multitape probabilistic Turing machine with k tapes and alphabet size β operating in time t and accepting the language $L(M)$ with error bounded by ϵ . Let ϵ' be a small positive constant such that $\epsilon + \epsilon' < 1/2$. We construct a probabilistic Turing machine M' accepting $L(M)$ with error bound $\epsilon + \epsilon'$ and operating in time $\text{poly}(t)$ that uses only $(1 - \delta')t + O((\log(t))^3) < (1 - \delta)t$ random bits, where δ' is a constant to be determined later, and δ is any positive constant less than δ' . Given an input x , M' begins by simulating M for $(1 - \delta')t$ steps. It stores the random bits r it uses on a separate tape. Let $C_r(x)$ be the configuration that M' attains after using the random bit sequence r . By Theorem 26, there is an explicit $(\delta't, \epsilon'/2)$ extractor $\text{Ext} : \{0, 1\}^{(1 - \delta')t} \times \{0, 1\}^{c(\log(t))^3} \rightarrow \{0, 1\}^{\delta't}$ computable in NC^1

and hence in polynomial time. M' applies Ext to arguments r and l for a random seed $l \in \{0, 1\}^{c(\log(t))^3}$ and continues the simulation of M from $C_r(x)$ on the string $Ext(r, l)$ instead of using a random string of length $s = \delta't$. M' operates in $poly(t)$ time and uses $(1 - \delta')t + O((\log(t))^3)$ random bits. We show below that, when δ' is chosen appropriately, M' has error bound $\epsilon + \epsilon'$ and accepts x iff M does.

The basic observation that we need is that when M is a multitape Turing machine, only ds bits of the configuration C_r are relevant to the last s steps of the computation of M , where $d = 2(k + 1) \log(\beta)$. If we think of the random string r as inducing a function from the last s random bits used by the computation to $\{0, 1\}$, the observation above means there can be at most 2^{ds} such functions. Let us denote the function induced by random string r by f_r . More precisely, given a random string r , let $f_r : \{0, 1\}^s \rightarrow \{0, 1\}$ be the function that maps a string r' to the outcome of M on the random string $R = rr'$. Think of the function f_r as a random variable on the sample space defined by r . Now we consider two cases for any fixed function $f : \{0, 1\}^s \rightarrow \{0, 1\}$ induced by some random string r . Either $f = f_r$ with probability at least $\epsilon'/(2^{ds+1})$ over the random strings r , or it does not. Let us call an f satisfying the first case a ‘‘probable’’ f and an f satisfying the second case an ‘‘improbable’’ f .

If f is probable, the distribution D_f on $\{0, 1\}^{(1-\delta')t}$, which is uniform on all strings r such that $f = f_r$, has high min-entropy, and we can approximate the probability (over the set of all its inputs) that f is 1 by extracting almost-random bits from D_f and evaluating f on the extracted bits. When f is improbable, we can make no guarantees about approximating the probability that f is 1, but by the observation in the previous paragraph, there are at most 2^{ds} such functions f , and we can bound the total error in this case. We now work out the argument in more detail.

Let p be the probability that M accepts on input x , and p' the probability that M' accepts on input x . We wish to bound the difference $|p - p'|$.

$$\begin{aligned}
(1) \quad |p - p'| &= \left| E_{R \in \{0, 1\}^t} [M(x, R)] - E_{R \in \{0, 1\}^{(1-\delta')t + c(\log(t))^3}} [M'(x, R)] \right| \\
(2) \quad &= \left| E_r [E_{r'} [f_r(r')]] - E_r \left[E_{l \in \{0, 1\}^{c(\log(t))^3}} [f_r(Ext(r, l))] \right] \right| \\
(3) \quad &\leq \sum_{f: \{0, 1\}^s \rightarrow \{0, 1\}} \Pr_r [f = f_r] \cdot |E_{r'} [f(r')] - E_{r \leftarrow D_{f, l}} [f(Ext(r, l))]| \\
&= \sum_{f \text{ probable}} \Pr_r [f = f_r] \cdot |E_{r'} [f(r')] - E_{r \leftarrow D_{f, l}} [f(Ext(r, l))]| \\
(4) \quad &+ \sum_{f \text{ improbable}} \Pr_r [f = f_r] \cdot |E_{r'} [f(r')] - E_{r \leftarrow D_{f, l}} [f(Ext(r, l))]| \\
(5) \quad &\leq \sum_{f \text{ probable}} \Pr_r [f = f_r] \cdot \epsilon'/2 + \sum_{f \text{ improbable}} \Pr_r [f = f_r] \cdot 1 \\
(6) \quad &\leq \frac{\epsilon'}{2} + \frac{2^{ds} \epsilon'}{(2^{ds+1})} \\
&= \epsilon'.
\end{aligned}$$

Equation (2) follows from (1) by splitting up the computations of M and M' into the first $(1 - \delta')t$ steps and the rest. Then (3) follows from (2) by grouping terms differently and applying the triangle inequality. (4) follows from (3) just by considering sums for probable and improbable f 's separately.

The second summation term in (5) follows from the second summation term in (4) because the difference in expectations of two $(0, 1)$ -random variables is bounded

above by 1. We describe how to choose δ' so that the first summation term in (4) is bounded by the first summation term in (5). If f is probable, the probability over r that r induces f is at least $\epsilon'/2^{d\delta't+1}$, and since D_f is uniform over all such r , D_f has min-entropy at least $(1-\delta')t - (d\delta't + \log(1/\epsilon') + 1)$. If we set $\delta' < 1/(d+2)$, D_f has min-entropy at least $\delta't$ for large enough t , and by assumption on the extractor Ext , the distribution $Ext(r, l)$, where r is chosen from D_f and l from $U_{c(\log(t))^3}$, is $\epsilon'/2$ -close to uniform. Hence, the distribution $f(Ext(r, l))$ is $\epsilon'/2$ -close to the distribution $f(r')$, where r' is chosen from the distribution $U_{\delta't}$, and the difference in expectations is bounded above by $\epsilon'/2$.

The first term in (6) follows from the first summation term in (5) because the events $f_r = f_1$ and $f_r = f_2$ are disjoint for distinct functions f_1, f_2 . To see that the second term in (6) follows from the second summation term in (5), note that there are at most 2^{ds} bad f 's, and for each of these the probability over r that r induces f is less than $\epsilon'/2^{ds+1}$.

Thus M' accepts L with error bound $\epsilon + \epsilon'$. By using the error reduction technique of [KPS85] (as in section 5), which does not use additional random bits, we obtain a simulating machine M' with the same error bound as the simulated machine. \square

Theorem 4 follows from Theorem 67 by running over all random strings of length $(1-\delta)t$. Using the efficient error reduction of [CW89], we obtain the following.

COROLLARY 68. *For any integers $k > 0$ and $\beta > 0$, there is a constant $\gamma > 0$ such that for each language L accepted by a probabilistic k -tape Turing machine with alphabet size β in time t with a constant error bound, L is accepted by a probabilistic multitape Turing machine using t random bits and running in time $\text{poly}(t)$ with error bound $2^{-\gamma t}$.*

Theorem 67 works in rather restricted situations. For one, the argument does not work for probabilistic classes defined by RAMs. The essential condition is that the machines defining the complexity class have polynomial ‘‘vicinity’’; i.e., the cardinality of the set of positions in any given configuration which can be accessed within s time steps is bounded above by a polynomial in s . This is true, for instance, of multi-dimensional Turing machines, but not for more general machines such as RAMs or pointer machines. Secondly, the argument works only when the probabilistic algorithm uses as much randomness as time. We can handle the situation where the number of computation steps between two consecutive accesses to the random tape is bounded, but not the general case in which the computation takes time linear in the number of random bits used, with a constant factor larger than 1. A further step would be deterministic simulations of randomized computations that run in time t and use r random bits, where the simulation runs in time $o(2^r) \cdot \text{poly}(t)$. However, as explained in section 1.3, this would imply new lower bounds in case $r = o(t)$.

Our approach works for randomized formulae and NC^1 circuits but not for general randomized circuits. Recall that a randomized circuit takes two inputs—a ‘‘random input’’ corresponding to the random bits used by the circuit and the ‘‘actual input,’’ such that for each setting of the actual input, the circuit either accepts with probability at least $2/3$ or rejects with probability at least $2/3$ over the choices of the random input.

The reason why our strategy fails for general circuits is that all but a logarithmic number of random inputs may need to be set on a given actual input in order to significantly reduce the size of the circuit. Formulae do not exhibit that problem. If we set all but the r^ϵ least popular variables occurring in a formula of size t on r variables, the resulting formula is of size no more than $s = t/r^{1-\epsilon}$. Since the number of formulae of size s on r^ϵ inputs is bounded by $2^{O(\epsilon s \log r)}$, we have the kind of

bound we need on the number of different future behaviors for our earlier argument, provided $s \log r$ is sufficiently less than r , or equivalently, that t is sufficiently less than $r^{2-\epsilon}/\log r$. Thus, we can simulate randomized formulae of less than quadratic size by picking the $r - r^\epsilon$ most popular random input bits uniformly at random, and extracting the r^ϵ remaining ones using the extractor from Theorem 26. Since that extractor is computable by a formula of polynomial size, the overall simulation can be implemented by a formula of polynomial size with r^ϵ fewer random input bits than the original formula.

Moreover, finding the r^ϵ least popular variables in a formula can be done by counting the number of occurrences of each variable, and sorting the variables using that count as the key. Since all these operations run in logspace (even in TC^0) and logspace computations compose, the overall transformation preserves logspace uniformity.

A similar argument works for randomized NC^1 circuits of restricted depth because of the standard conversion between NC^1 circuits of depth d and formulae of size $2^{\Theta(d)}$. We now provide more details for that case.

THEOREM 69. *For each constant $\epsilon > 0$, for each LOGSPACE-uniform family of randomized NC^1 circuits using r random bits of depth $(2 - \epsilon) \log(r)$, there is an equivalent LOGSPACE-uniform family of randomized NC^1 circuits using $r - r^{\epsilon/2}$ random bits.*

Proof. Let $\{C_n\}$ be a family of randomized NC^1 circuits taking a random input of length r and having depth $(2 - \epsilon) \log(r)$. We first describe how to construct an equivalent circuit family $\{C'_n\}$ that uses less randomness, and then explain how the uniformity of the circuit family can be preserved.

Fix an integer n . The idea of the simulation is to set all but $r^{\epsilon/2}$ of the random input variables of C_n to random bits, and then extract values for the remaining $r^{\epsilon/2}$ random bits using the extractors from Theorem 26. To argue that this works, we consider a representation of C_n which is semantically equivalent but easier to work with, namely a formula. By standard techniques, C_n can be transformed into a formula F_n of size $2^{(2-\epsilon) \log(r)} = r^{2-\epsilon}$. This formula contains variables for both the random input bits and the actual input bits. Let m be the number of occurrences of actual input variables in the formula. Now we need to choose values for some of the random input variables of the formula so that the formula decreases in size by a polynomial fraction when those variables are set. We do this by ranking the random input variables in decreasing order of frequency of occurrence in the formula F_n and choosing the $r^{\epsilon/2}$ least frequently occurring variables to be left unset. Setting the other random input variables and the actual input variables always reduces the size of the formula to $r^{\epsilon/2}(r^{2-\epsilon} - m)/r \leq r^{1-\epsilon/2}$ or less. Thus, as explained in the sketch before the proof, the number of different residual formulae is small enough to carry through an argument similar to the one in the proof of Theorem 67—the extractor from Theorem 26 can be successfully applied to recycle the randomness of the set random input variables and estimate the probability that the residual formula is 1.

Since the extractor of Theorem 26 can be evaluated in logarithmic depth, it is clear that the new circuit C'_n has logarithmic depth. C'_n has a slightly larger error bound than C_n , but this can be reduced by a constant factor by doing expander-based amplification [BYGW99].

In order to argue the logspace uniformity of the resulting family $\{C'_n\}$, the critical step is determining the $r^{\epsilon/2}$ least popular random input variables in F_n . Note that the transformation from C_n into F_n can be performed in logspace. Once we have F_n ,

we can follow the strategy outlined before: Count the number of occurrences of each random input variable, rank the variables in increasing order of that count, and select the first $r^{\epsilon/2}$. The overall procedure runs in logspace since each of its components does. \square

Finally, we wish to draw attention to the fact that the proof of Theorem 69 is a rare example of a “non-black-box” technique; i.e., the deterministic simulation does not merely query the results of executing the probabilistic circuit on certain random strings but also takes the structure of the circuit into consideration. More sophisticated use of these techniques may even yield simulations that beat oracle constructions. But the specific technique used here does not seem capable of reducing the amount of randomness by more than a factor of $1/2$. This is because the number of future behaviors on the last s random bits is typically at least 2^s , which implies that we need to extract s bits of randomness from the first $r - s$ random bits. The latter is impossible if $s > r/2$. Iterated applications of the idea seem difficult because the bound of $2^{O(s)}$ on the number of future behaviors typically does not hold for the *next* s steps but only for the *last* s .

Acknowledgments. Thanks to Eric Allender for his comments on an earlier version of this paper. Thanks also to Janos Simon for comments on an earlier version, and for his encouragement. We are grateful to Prahlahd Harsha and Shien Jin Ong for pointing out a problem in an earlier version of this paper. We also want to thank Albert Atserias and Michal Koucky for interesting discussions, and the anonymous referees for their thorough reports and helpful suggestions.

REFERENCES

- [AG91] E. ALLENDER AND V. GORE, *On strong separations from AC^0* , Fundamentals of Computation Theory, 8 (1991), pp. 1–15.
- [Ajt83] M. AJTAI, Σ_1^1 -formulae on finite structures, Ann. Pure Appl. Logic, 24 (1983), pp. 1–48.
- [AKS02] M. AGRAWAL, N. KAYAL, AND N. SAXENA, *PRIMES is in P*, Technical report, Department of Computer Science and Engineering, Indian Institute of Technology, Kanpur, India, 2002.
- [BDG88] J. L. BALCÁZAR, J. DÍAZ, AND J. GABARRÓ, *Structural Complexity 1*, Springer-Verlag, New York, 1988.
- [BDG90] J. L. BALCÁZAR, J. DÍAZ, AND J. GABARRÓ, *Structural Complexity 2*, Springer-Verlag, New York, 1990.
- [BFL91] L. BABAI, L. FORTNOW, AND C. LUND, *Non-deterministic exponential time has two-prover interactive protocols*, Comput. Complexity, 1 (1991), pp. 3–40.
- [BFLS91] L. BABAI, L. FORTNOW, L. A. LEVIN, AND M. SZEGEDY, *Checking computations in polylogarithmic time*, in Proceedings of the 23rd Annual ACM Symposium on Theory of Computing, New Orleans, LA, 1991, ACM, New York, pp. 21–32.
- [BFNW93] L. BABAI, L. FORTNOW, N. NISAN, AND A. WIGDERSON, *BPP has subexponential time simulations unless EXPTIME has publishable proofs*, Comput. Complexity, 3 (1993), pp. 307–318.
- [BM84] M. BLUM AND S. MICALI, *How to generate cryptographically strong sequence of pseudo-random bits*, SIAM J. Comput., 13 (1984), pp. 850–864.
- [BM88] L. BABAI AND S. MORAN, *Arthur-Merlin games: A randomized proof system, and a hierarchy of complexity classes*, J. Comput. System Sci., 36 (1988), pp. 254–276.
- [BYGW99] Z. BAR-YOSSEF, O. GOLDREICH, AND A. WIGDERSON, *Deterministic amplification of space-bounded probabilistic algorithms*, in Proceedings of the 14th Annual IEEE Conference on Computational Complexity, Los Alamitos, NV, 1999, IEEE Computer Society, Piscataway, NJ, pp. 188–199.
- [CW89] A. COHEN AND A. WIGDERSON, *Dispersers, deterministic amplification, and weak random sources*, in Proceedings of the 30th Annual IEEE Symposium on Foundations of Computer Science, Chapel Hill, NC, 1989, IEEE Press, Piscataway, NJ, pp. 14–19.

- [FSS84] M. FURST, J. B. SAXE, AND M. SIPSER, *Parity, circuits, and the polynomial-time hierarchy*, Math. Systems Theory, 17 (1984), pp. 13–27.
- [GHS91] J. G. GESKE, D. T. HUYNH, AND J. I. SEIFERAS, *A note on almost-everywhere-complex sets and separating deterministic-time-complexity classes*, Inform. and Comput., 92 (1991), pp. 97–104.
- [Gol00] O. GOLDBREICH, *Pseudorandomness*, in Annual International Colloquium on Automata, Languages and Programming, 2000, pp. 687–704.
- [GW02] O. GOLDBREICH AND A. WIGDERSON, *Derandomization that is rarely wrong from short advice that is typically good*, in Proceedings of the 6th International Workshop on Randomization and Approximation Techniques in Computer Science, Cambridge, MA, 2002, pp. 209–223.
- [Häs86] J. HÄSTAD, *Almost optimal lower bounds for small depth circuits*, in Proceedings of the 18th Annual ACM Symposium on Theory of Computing, Berkeley, CA, 1986, ACM, New York, pp. 6–20.
- [Hel86] H. HELLER, *On relativized exponential and probabilistic complexity classes*, Inform. and Control, 71 (1986), pp. 231–243.
- [HR00] T. HARTMAN AND R. RAZ, *On the distribution of the number of roots of polynomials and explicit logspace extractors*, in Proceedings of the 4th International Workshop on Randomization and Approximation Techniques in Computer Science, Geneva, Switzerland, 2000, pp. 3–22.
- [IKW02] R. IMPAGLIAZZO, V. KABANETS, AND A. WIGDERSON, *In search of an easy witness: Exponential time vs. probabilistic polynomial time*, J. Comput. System Sci., 65 (2002), pp. 672–694.
- [INW94] R. IMPAGLIAZZO, N. NISAN, AND A. WIGDERSON, *Pseudorandomness for network algorithms*, in Proceedings of the 26th Annual ACM Symposium on the Theory of Computing, Montreal, QC, 1994, ACM, New York, pp. 356–364.
- [IW97] R. IMPAGLIAZZO AND A. WIGDERSON, *$P = BPP$ if E requires exponential circuits: Derandomizing the XOR lemma*, in Proceedings of the 29th Annual ACM Symposium on the Theory of Computing, El Paso, TX, 1997, ACM, New York, pp. 220–229.
- [IW01] R. IMPAGLIAZZO AND A. WIGDERSON, *Randomness vs. time: Derandomization under a uniform assumption*, J. Comput. System Sci., 63 (2001), pp. 672–688.
- [IZ89] R. IMPAGLIAZZO AND D. ZUCKERMAN, *How to recycle random bits*, in Proceedings of the 30th Annual IEEE Symposium on Foundations of Computer Science, Research Triangle Park, NC, 1989, IEEE Press, Piscataway, NJ, pp. 248–253.
- [Kab01] V. KABANETS, *Easiness assumptions and hardness tests: Trading time for zero error*, J. Comput. System Sci., 63 (2001), pp. 236–252.
- [KI03] V. KABANETS AND R. IMPAGLIAZZO, *Derandomizing polynomial identity tests means proving circuit lower bounds*, J. Comput. Complexity, 13 (2004), pp. 1–46.
- [KL82] R. KARP AND R. LIPTON, *Turing machines that take advice*, L’Enseignement Mathématique, 28 (1982), pp. 191–209.
- [KPS85] R. KARP, N. PIPPENGER, AND M. SIPSER, *A time randomness tradeoff*, in Proceedings of the AMS Conference on Probabilistic Computational Complexity, Durham, NH, 1985.
- [KV96] M. KARPINSKI AND R. VERBEEK, *On randomized versus deterministic computation*, Theoret. Comput. Sci., 154 (1996), pp. 23–39.
- [KvM02] A. R. KLIVANS AND D. VAN MELKEBEEK, *Graph nonisomorphism has subexponential size proofs unless the polynomial hierarchy collapses*, SIAM J. Comput., 31 (2002), pp. 1501–1526.
- [Lu01] C.-J. LU, *Derandomizing Arthur-Merlin games under uniform assumptions*, Comput. Complexity, 10 (2001), pp. 247–259.
- [NW94] N. NISAN AND A. WIGDERSON, *Hardness vs. randomness*, J. Comput. System Sci., 49 (1994), pp. 149–167.
- [NZ96] N. NISAN AND D. ZUCKERMAN, *Randomness is linear in space*, J. Comput. System Sci., 52 (1996), pp. 43–52.
- [Sha02] R. SHALTIEL, *Recent developments in explicit constructions of extractors*, Bull. Eur. Assoc. Theor. Comput. Sci. EATCS, 77 (2002), pp. 67–95.
- [Sip88] M. SIPSER, *Expanders, randomness, or time versus space*, J. Comput. System Sci., 36 (1988), pp. 379–383.
- [SSZ98] M. SAKS, A. SRINIVASAN, AND S. ZHOU, *Explicit OR-dispersers with polylogarithmic degree*, J. ACM, 45 (1998), pp. 123–154.

- [SU01] R. SHALTIEL AND C. UMANS, *Simple extractors for all min-entropies and a new pseudo-random generator*, in Proceedings of the 42nd IEEE Symposium on Foundations of Computer Science, Las Vegas, NV, 2001, IEEE Press, Piscataway, NJ, pp. 648–657.
- [SvM03] R. SANTHANAM AND D. VAN MELKEBEEK, *Holographic proofs and derandomization*, in Proceedings of the 18th Annual IEEE Conference on Computational Complexity, Aarhus, Denmark, 2003, IEEE Press, Piscataway, NJ, pp. 269–283.
- [vL98] J. H. VAN LINT, *Introduction to Coding Theory*, Springer-Verlag, Berlin, 1998.
- [Yao82] A. YAO, *Theory and application of trapdoor functions*, in Proceedings of the 23rd Annual IEEE Symposium on Foundations of Computer Science, Chicago, 1982, IEEE Press, Piscataway, NJ, pp. 80–91.

APPROXIMATING THE WEIGHT OF THE EUCLIDEAN MINIMUM SPANNING TREE IN SUBLINEAR TIME*

ARTUR CZUMAJ[†], FUNDA ERGÜN[‡], LANCE FORTNOW[§], AVNER MAGEN[¶],
ILAN NEWMAN^{||}, RONITT RUBINFELD[#], AND CHRISTIAN SOHLER^{††}

Abstract. We consider the problem of computing the weight of a Euclidean minimum spanning tree for a set of n points in \mathbb{R}^d . We focus on the setting where the input point set is supported by certain basic (and commonly used) geometric data structures that can provide efficient access to the input in a structured way. We present an algorithm that estimates with high probability the weight of a Euclidean minimum spanning tree of a set of points to within $1 + \varepsilon$ using only $\tilde{O}(\sqrt{n} \text{poly}(1/\varepsilon))$ queries for constant d . The algorithm assumes that the input is supported by a minimal bounding cube enclosing it, by orthogonal range queries, and by cone approximate nearest neighbor queries.

Key words. sublinear algorithms, minimum spanning tree

AMS subject classifications. 68W25, 68W20

DOI. 10.1137/S0097539703435297

1. Introduction. As the power and the connectivity of computers increase and the cost of memory becomes cheaper, we have become inundated with large amounts of data. Although traditionally linear time algorithms were sought to solve our problems, it is no longer clear that a linear time algorithm is good enough in every setting. The question then is whether we can solve *anything* of interest in sublinear time, when the algorithm is not even given time to read all of the input data. The answer is yes; in recent years, several sublinear time algorithms have been presented that solve a wide range of property testing and approximation problems.

In this paper we consider the problem of estimating the weight of a minimum spanning tree, where the input is a set of points in the Euclidean space \mathbb{R}^d . Since the location of a single point may dramatically influence the value of the weight of the Euclidean minimum spanning tree (EMST), we cannot hope to get a reasonable

*Received by the editors September 26, 2003; accepted for publication (in revised form) December 1, 2004; published electronically September 8, 2005. A preliminary version of this paper appeared in *Proceedings of the 14th Annual ACM-SIAM Symposium on Discrete Algorithms*, Baltimore, MD, 2003, ACM, New York, SIAM, Philadelphia, pp. 813–822.

<http://www.siam.org/journals/sicomp/35-1/43529.html>

[†]Department of Computer Science, New Jersey Institute of Technology, Newark, NJ 07102 (czumaj@cis.njit.edu). This author's research was supported in part by NSF grant CCR-0105701.

[‡]School of Computer Science, Simon Fraser University, Vancouver, BC V5A 1S6, Canada (funda@cs.sfu.ca). Part of this work was done while the author was at NEC Research, Princeton, NJ 08540.

[§]Department of Computer Science, University of Chicago, Chicago, IL 60637 (fortnow@cs.uchicago.edu). Part of this work was done while the author was at NEC Research, Princeton, NJ 08540.

[¶]Department of Computer Science, University of Toronto, Toronto, ON M5S 3G4, Canada (avner@cs.toronto.edu). Part of this work was done while the author was at NEC Research, Princeton, NJ 08540.

^{||}Department of Computer Science, University of Haifa, Haifa, Israel (ilan@cs.haifa.ac.il). Part of this work was done while the author was at NEC Research, Princeton, NJ 08540.

[#]MIT CSAIL, Cambridge, MA 02139 (ronitt@theory.csail.mit.edu). Part of this work was done while the author was at NEC Research, Princeton, NJ 08540.

^{††}Heinz Nixdorf Institute and Faculty of Computer Science, Electrical Engineering and Mathematics, University of Paderborn, D-33102 Paderborn, Germany (csohler@uni-paderborn.de). This author's research was partly supported by DFG grant Me 872/8-1 and by EU grant IST-1999-14186 (ALCOM-FT).

approximation in sublinear time with access only to the locations of the points. This is true even when we consider probabilistic algorithms. However, it is often the case that massive databases, particularly in a geometric context, contain sophisticated data structures on top of the raw data that support various forms of queries. Examples of such queries are the nearest neighbor of a point, or the point with the highest value in a coordinate. Consequently, in this paper, we assume that algorithms have access to certain commonly used data structures which aid the algorithm in its computation. This may be considered a motivation for maintaining such data structures, particularly if they aid in other tasks as well.

1.1. Results. In this paper we describe three algorithms for estimating the weight of an EMST over n given points in a Euclidean space \mathbb{R}^d , where the algorithms are given access to basic geometric data structures supporting the input. Throughout the paper we assume that d is a constant, though our analysis can be easily carried over for arbitrary values of d . It should be noted that our algorithms do not supply a low weight spanning tree (which takes linear space to represent) but only estimate its weight.

We first consider the case when the algorithm is given, in addition to access to the input point set, (1) a *minimal bounding cube* that contains all points in the input set and (2) access to an *orthogonal range query* data structure which, given an axis-parallel cube, answers whether there is an input point within the cube. In this model, we give a deterministic $\mathcal{O}(n^{1/2})$ -time algorithm for the two-dimensional case that outputs a value w such that $\frac{1}{\alpha} \text{EMST}(P) - \mathcal{L}n^{-c} \leq w \leq \alpha \text{EMST}(P) + \mathcal{L}n^{-c}$, where $\alpha = \Theta(n^{1/8} \log n)$, \mathcal{L} is the side length of a minimal axis parallel bounding cube of the point set, and c is an arbitrary constant. We also show that any deterministic algorithm that uses $o(n^{1/2})$ orthogonal range queries cannot significantly improve the quality of approximation.

We next consider the case where, in addition to the above data structures, we are also given (3) access to a *cone nearest neighbor* data structure, which, given a point p and a cone C , returns a nearest point to p in the cone $p + C$. Our second algorithm combines the extra power of the cone nearest neighbor data structures with ideas from the recent randomized sublinear-time algorithm for estimating the minimum spanning tree (MST) in general graphs [10]. The algorithm outputs a value which, with high probability, is within a $1 + \varepsilon$ factor of the EMST, and it runs in $\mathcal{O}(\Lambda/\varepsilon^3)$ time, where Λ is the *spread* of P (the ratio of the maximum and minimum distances between points in P); observe that Λ can be arbitrarily large.

Our main contribution is the third algorithm that does not have any dependency on Λ and requires only cone *approximate* nearest neighbor queries, which we define in the next section. For a constant d , the algorithm runs in $\tilde{\mathcal{O}}(\sqrt{n} \text{poly}(1/\varepsilon))$ time and outputs an approximation of the EMST weight to within a multiplicative factor of $1 + \varepsilon$ with high probability. The algorithm combines the ideas from our first two algorithms. It partitions the input points into components and estimates the EMST separately by considering pairs of points that lie in the same component and pairs of points that belong to different components. To estimate the EMST within components, we use an extension of our second algorithm. To estimate the weight required to connect the components we use a variant of our first algorithm. The combination of these two algorithms leads to a significant improvement in the quality of approximation (compared to the first algorithm) and in the running time (compared to the second algorithm).

We notice also that our algorithms lead to sublinear-time $(2 + \varepsilon)$ -approximation

algorithms for two other classical geometric problems: Euclidean TSP and the Euclidean Steiner tree problem. These results follow from the well-known relationship between the weight of EMST and the weight of the Euclidean TSP and of the Euclidean Steiner tree (see, e.g., [24]). Indeed, it is known that in metric spaces the weight of the Euclidean TSP is between the weight of the EMST and twice the EMST weight. Similarly, it is known that in metric spaces the EMST weight is between the weight of the Steiner tree and twice its weight. On the plane, one can improve this result by using the fact that the EMST weight is upper bounded by at most $2/\sqrt{3}$ times the weight of the Euclidean Steiner tree [13].

1.2. Relation to previous works. The EMST problem is a classical problem in computational geometry and has been extensively studied in the literature for more than two decades. It is easy to see that to find the EMST of n points, $\mathcal{O}(dn^2)$ time suffices, by reducing the problem to finding the MST in dense graphs. In the simplest case where $d = 2$ (on the plane), Shamos and Hoey [22] show that the EMST problem can be solved in $\mathcal{O}(n \log n)$ time. For $d \geq 3$, no $\tilde{\mathcal{O}}(n)$ -time algorithm is known, and it is a major open question whether an $\mathcal{O}(n \log n)$ -time algorithm exists even for $d = 3$ [16]; in fact, it is even conjectured (see, e.g., [16]) that no $o(n^{4/3})$ -time algorithm does exist. Yao [26] was the first who broke the $\mathcal{O}(n^2)$ -time barrier for $d \geq 3$ and designed an $\tilde{\mathcal{O}}(n^{1.8})$ -time algorithm for $d = 3$. This bound has since been improved, and the fastest currently known (randomized) algorithm achieves the running time of $\tilde{\mathcal{O}}(n^{4/3})$ [2] for $d = 3$ (and the running time tends to $\mathcal{O}(n^2)$ as d grows). Significantly better bounds can be achieved if one allows the output to be approximated. Callahan and Kosaraju [7] give a $\mathcal{O}(n \log n + n \log(1/\varepsilon) \varepsilon^{-d/2})$ -time algorithm that finds an approximate EMST to within a multiplicative factor of $1 + \varepsilon$.

Our algorithms rely on a recent randomized algorithm of [10] that, given a connected graph in adjacency list representation with average degree d , edge weights in the range $[1, \dots, W]$, and a parameter $0 < \varepsilon < \frac{1}{2}$, approximates, with high probability, the weight of an MST in time $\tilde{\mathcal{O}}(dW \varepsilon^{-3})$ within a factor of $1 + \varepsilon$. The time bound does not directly depend on the number of vertices or edges in the graph. We emphasize, however, that our representation is quite different, and in general would give a graph with average degree n . Therefore, a direct application of this result to the EMST problem does not lead to a sublinear-time algorithm.

We also note that very recently a similar approach which also partitions the input points on a grid similar to that used in our paper has been used by Indyk [18] in the problem of approximating the EMST in the streaming model in one pass. He showed that the EMST of n points from $\{1, \dots, \Delta\}^d$ is approximated to within an $\mathcal{O}(d \log \Delta)$ factor using $\mathcal{O}(d \log^2 \Delta)$ bits of storage for dynamic data streams through hierarchically well-separated trees defined over a set of grids over the input points. In another recent work, Czumaj and Sohler [12] designed a randomized $(1 + \varepsilon)$ -approximation algorithm for metric MST with the running time $\mathcal{O}(n \text{polylog}(n)/\varepsilon^{\mathcal{O}(1)})$, where the input graph has n vertices and $\Theta(n^2)$ edges.

1.3. Dynamic algorithms. Our model of computation is also interesting in the context of dynamic algorithms. There exist fully dynamic algorithms that maintain the EMST subject to point insertions and deletions; [15] gives an algorithm with amortized time $\tilde{\mathcal{O}}(\sqrt{n})$ and $\mathcal{O}(n^{1-\varepsilon})$ per update operation for $d \leq 4$ and $d > 4$, respectively. A disadvantage of this algorithm (and of all typical dynamic algorithms) is that it requires as much as $\tilde{\mathcal{O}}(\sqrt{n})$ time per input update, making the algorithm very costly in situations where the EMST queries are very rare. The data structures we require

in our setting are dynamically maintained by standard geometric databases anyway. Thus, if the database supports all required data structures in polylogarithmic time, the amortized time required by our algorithm is $\tilde{O}(\sqrt{n}/U)$, where U is the typical number of updates per one EMST calculation. We note again that our algorithm does not supply the MST but returns only its approximate weight.

1.4. Discussion on the model of computations. Our results assume that the data is organized using certain types of commonly used data structures. This idea has been frequently used in other areas such as databases, where one assumes that a selected set of operations on the top of the data set is provided to the user. A similar model of computations has been used recently in a related context of geometric computations in [11] (see also [23]).

The data structures used in our paper are commonly used in algorithms in general, and some (as in supporting approximate nearest neighbor queries) are used more in the context of computational geometry. Our algorithm cannot be expected to construct these data structures given its sublinear nature. Neither can it be assumed to have collected the input set. The assumption that some other source has collected the data is inherent in the model of sublinear algorithms, and in our paper we strengthen this assumption to require that during this preprocessing, some well-known and natural data structures have been built. In most of the cases, the construction time for these data structures is not much in excess of the time needed to read and store the entire data set. One can view this as having separate agents performing the data collection and computation. Alternately, one can think of the data collection and organization step as preprocessing, to be done only once. One would like to know the degree of dependence on these underlying data structures and explore the tradeoff resulting from them. We believe that this work is an important step in understanding the nature of geometric problems and the time/resource tradeoffs (resources can include the underlying data structures) involved in their solutions. We would like to learn what inherent properties of a problem and its data set lead to better sublinear algorithms. In the future we expect to solve more problems in this area and develop an understanding of the inherent structure of geometric objects and their collections that allow different time/resource tradeoffs.

Organization of the paper. We start by presenting an algorithm that needs access to only a minimal bounding cube of the point set P and to an orthogonal range query oracle in section 3. In section 5, we present a simple algorithm that uses additionally the cone nearest neighbor oracle. Finally, in section 6, we discuss the main contribution of this paper, a sublinear time algorithm that uses a minimal bounding cube oracle, the orthogonal range query oracle, and the cone *approximate* nearest neighbor oracle.

2. Preliminaries. For a given set P of points in a Euclidean space \mathbb{R}^d , a (*Euclidean*) *graph* on P can be modeled as a weighted undirected graph $G = (P, E)$, where P is a vertex set, E is a subset of the (unordered) pairs of points in P , and the *length/weight* of edge $\{p, q\}$ is equal to the Euclidean distance between points p and q , denoted $|p - q|$. The *weight of the graph* is the sum of the weights of its edges.

Throughout the paper we denote by \mathbb{K}_P the complete (undirected) graph on P where the edge weights are the Euclidean distances between the endpoints. A graph G on a set of points P is called a *Euclidean minimum spanning tree* (EMST) of P if it is a minimum-weight spanning subgraph of \mathbb{K}_P . We denote by $\text{EMST}(P)$ both the EMST of P and the weight of the EMST of P . Similarly, for a given graph G we

will denote by $\text{MST}(G)$ the minimum spanning tree of G as well as the weight of the minimum spanning tree of G .

For a given point set P , we denote by Λ the *spread* of P , that is, the ratio of the maximum and the minimum distances between points in P . We let \mathcal{BC} be a minimal bounding cube of P (which is made available via the *minimal bounding cube oracle*) and let \mathcal{L} denote its side length.

2.1. Models of computation. In this paper we use some basic geometric data structures supporting access to the input point set. Given a point set P in \mathbb{R}^d , we use data structures supporting the following types of queries:

(i) *minimal bounding cube of P* : returns the location of a minimum size axis-parallel d -dimensional cube containing P ; that is, it returns the location of a cube $C = [a_1, a_1 + \mathcal{L}] \times [a_2, a_2 + \mathcal{L}] \times \cdots \times [a_d, a_d + \mathcal{L}]$ that contains P such that no axis-parallel cube of edge length smaller than \mathcal{L} contains P ;

(ii) *(orthogonal) range query oracle*: for a given axis-parallel cube C , tests whether C contains a point from P ;

(iii) *cone $(1 + \delta)$ -approximate nearest neighbor oracle*: δ is any nonnegative real number, and it is assumed that a set of cones \mathcal{C} with apexes at the origin is given in advance. The cone $(1 + \delta)$ -approximate nearest neighbor oracle, for a given point $p \in P$ and a given cone $C \in \mathcal{C}$, returns a $(1 + \delta)$ -approximate nearest neighbor¹ of p in $(P \setminus \{p\}) \cap (p + C)$. (We denote by $p + C$ the translated cone $\{a + p : a \in C\}$.) If $(P \setminus \{p\}) \cap (p + C)$ is empty, then a special value is returned.

In the special case where $\delta = 0$, the oracle gives the true nearest neighbor and is simply called the *cone nearest neighbor oracle*.

2.1.1. Implementing supporting data structures. To make our model of computations viable, we discuss here how our supporting data structures (oracles) can be implemented efficiently using standard geometric data structures.

Minimal bounding cube. The query about the *minimal bounding cube* of a set of points $P \in \mathbb{R}^d$ can be supported by many standard geometric data structures. Indeed, the only information required to find the minimal bounding cube is to know the minimum and maximum d -dimensional coordinates of all input points. Therefore, many standard geometric data structures can support this query in time $\mathcal{O}(d)$ or $\mathcal{O}(d \log n)$.

Orthogonal range query oracle. There are many efficient data structures supporting the *orthogonal range query oracle*, and actually, orthogonal range queries are perhaps the most widely supported geometric queries (for survey expositions, see, e.g., [1, 3, 6]). One of the first data structures for orthogonal range searching is the *quad-tree*. Despite its bad worst-case behavior, the quad-tree is still used in many applications because it provides an easy-to-implement linear-space data structure that often has a very good performance. Another standard data structure for orthogonal range queries is the range tree. It can be used to report the k points in a query range in $O(\log^d n + k)$ time. Using the fractional cascading technique, the query time can be improved to $O(\log^{d-1} n + k)$ [20, 25]. The best known data structures for orthogonal range searching based on compressed range trees and some other techniques such as filtering search can be found in [8, 9]. For more details we refer to the survey [3].

Cone nearest neighbor oracle. In the seminal paper on Euclidean minimum spanning trees, Yao [26] examined algorithms for cone nearest neighbor in the cones with

¹For a point $p \in P$ and a set of points $Q \subseteq \mathbb{R}^d$, a $(1 + \delta)$ -approximate nearest neighbor of p in Q is any point $q \in Q$ such that for every $x \in Q$ it holds that $|p - q| \leq (1 + \delta) \cdot |p - x|$.

the angular diameter $\pi/4$. Cone nearest neighbor queries have also been studied extensively in follow-up papers dealing with the EMST problem (see, e.g., [2]).

Cone approximate nearest neighbor oracle. Cone *approximate* nearest neighbor queries have been widely investigated. They play an important role in the context of construction of Euclidean spanners (see, e.g., [4, 5, 14, 21]). And thus, among others, Ruppert and Seidel [21] show how to answer a query in amortized time $\mathcal{O}(n \log^{d-1} n)$ per cone in C , simultaneously for all points in P ; a similar construction is presented in [5]. Arya, Mount, and Smid [4] present a fully dynamic algorithm which, in polylogarithmic time, supports cone approximate nearest neighbor queries. Notice also that a single cone approximate nearest neighbor query can be answered using a logarithmic number of *simplex (triangular) range queries*, which is another classical geometric data structure (see, e.g., [1, 3, 6]).

3. Estimating the EMST with bounding cube and range queries. In this section we describe a natural approach to the approximation of $\text{EMST}(P)$ using minimum bounding cube oracle and orthogonal range queries. This approach by itself does not give a good enough multiplicative approximation but is used as a building block in the sublinear algorithm we present later. For simplicity, we describe in detail only the two-dimensional case ($d = 2$); the algorithm can be generalized to arbitrary d in an obvious way. The algorithm we supply is deterministic and outputs a value w such that $\frac{1}{\alpha} \text{EMST}(P) - \beta \leq w \leq \alpha \text{EMST}(P) + \beta$, where $\alpha = \mathcal{O}(n^{1/8} \log n)$, and $\beta = \mathcal{L} n^{-c}$, where \mathcal{L} is the side length of a minimal bounding cube of P and c is a constant. The algorithm has a running time of $\mathcal{O}(n^{1/2})$. We also show that any algorithm that uses the same running time (in fact, the same amount of queries and arbitrarily large running time) cannot significantly improve the quality of the approximation.

3.1. The quad-tree algorithm. We apply a standard quad-tree subdivision to the bounding cube \mathcal{BC} (see, e.g., [6, Chapter 14]). That is, we first partition \mathcal{BC} into four disjoint blocks (squares) of equal size. We can check which blocks contain points from P via orthogonal range queries. We then further subdivide the nonempty blocks and iterate this process as long as fewer than \sqrt{n} queries are made. This induces a tree structure on the blocks, where a block at level i has side length $\mathcal{L}/2^i$. Let k be the depth of this tree. We may assume that all nonempty blocks at level $k-1$ were subdivided into subblocks (of level k) and each subblock of level k was queried. Let B be the set of nonempty blocks at level k and let $b = |B|$. Clearly $b = \mathcal{O}(\sqrt{n})$. We now run any MST algorithm (as we will see later, a $(1 + \varepsilon)$ -approximation is good enough) on the centers of the blocks in B . This would result in a value L . We set $U = L + s\sqrt{bn}$, where $s = \mathcal{L} \cdot 2^{-k}$, and output the value $w = \sqrt{LU}$ as an approximation for $T^* = \text{EMST}(P)$.

CLAIM 1. *For an arbitrary constant c , $\frac{1}{\alpha} T^* - \beta \leq w \leq \alpha T^* + \beta$, where $\alpha = \mathcal{O}(n^{1/4} \log n)$ and $\beta = \mathcal{L} n^{-c}$.*

Proof. First note that the MST of any n points in a d -dimensional cube with side length h is $\mathcal{O}(h n^{\frac{d-1}{d}})$ and that this bound is tight (i.e., it is achievable for some inputs); see, e.g., [19]. Now, we let L^* be the weight of a minimum weight tree that touches every block in B . It is easy to see that $L^* \leq T^* \leq U$ (the last inequality is by the above upper bound and by using convexity).

Assume now that $b \geq \sqrt{n}/(4(c+1) \log n)$; then it can be seen that L upper bounds L^* and approximates it within an additive term of $\mathcal{O}(sb)$, and hence within a constant factor, say δ . Namely, $a \cdot b \cdot s \leq L^* \leq L \leq \delta \cdot L^*$ for some constants a and δ .

Hence, as U is an upper bound on T^* , the approximation factor turns out to be

$\alpha = \max\{U/w, w/L^*\}$. By our choice of w and the fact that L approximates L^* up to a constant, we get $\alpha = \mathcal{O}(\frac{U}{w}) = \mathcal{O}(\sqrt{\frac{U}{L^*}}) = \mathcal{O}((\frac{s\sqrt{bn}}{L})^{1/2}) = \mathcal{O}((n/b)^{1/4})$ (where the last inequality follows by plugging in the expression for U and L , and the previous follows from the fact that L approximates L^* within a constant factor). Now, by the above bound on L and on b , we obtain that $\alpha \leq \tilde{\mathcal{O}}(n^{1/8})$. Note that if we used an approximation L' guaranteed to be within a constant factor of L , we would still get the same result.

Assume now that $b < \sqrt{n}/(4(c+1)\log n)$. Then it can be seen that the depth of the quad-tree is at least $(c+1)\log n$ and hence $s \leq \mathcal{L} \cdot n^{-(c+1)}$. Therefore, the additive term is upper bounded by $U - L \leq \mathcal{O}(s \cdot \sqrt{bn}) = \mathcal{O}(n^{-(c+1)} \cdot \mathcal{L} \cdot n) = \mathcal{O}(\mathcal{L} \cdot n^{-c})$. \square

A note on the running time is due here. We use $\mathcal{O}(\sqrt{n})$ queries in the course of constructing the quad-tree. Next, we have to find the MST (or any $(1+\delta)$ approximation to it for any fixed δ). In the two-dimensional case this can be done in $\tilde{\mathcal{O}}(\sqrt{n})$ time [22], and this term dominates the total complexity.

Higher dimensions. In the case of dimension $d > 2$ the quad-tree has to be replaced with a 2^d -ary tree. The algorithm will be run similarly to the above until $\mathcal{O}(2^d \sqrt{n})$ queries have been made and all rectangles at the bottom level have been queried. Then, L is set similarly to the two-dimensional case, and $U = L + s \cdot n^{\frac{d-1}{d}} \cdot b^{1/d}$. The approximation w for T^* is taken to be the same. To have an efficient running time, a constant approximation for L can be used, rather than the exact value. This can be done in time $\mathcal{O}(n \log n)$ by the result of Callahan and Kosaraju [7].

It is easy to see that the following replaces Claim 1 with an analogous proof.

CLAIM 2. *For an arbitrary constant c , $\frac{1}{\alpha}T^* - \beta \leq w \leq \alpha T^* + \beta$, where $\alpha = \mathcal{O}(2^{d/2} \cdot n^{(d-1)/4d} \log n)$ and $\beta = \mathcal{L} n^{-c}$.*

As it turns out, the above quality of approximation is nearly optimal for the given time bound as shown by the following claim (shown only for the two-dimensional case; a similar result is true for the d -dimensional case as well).

CLAIM 3. *Any deterministic algorithm for approximating $\text{EMST}(P)$ in the two-dimensional case that uses $\mathcal{O}(\sqrt{n})$ orthogonal range queries has an approximation factor of $\Omega(n^{1/4})$.*

Proof. Consider any deterministic algorithm that uses at most \sqrt{n} range queries. Consider the following adversary for supplying the answer to the queries: The adversary will subdivide the unit square into a mesh of squares, each of side length $s = \frac{n^{-1/4}}{10}$, namely, into $100\sqrt{n}$ squares, denoted blocks. The adversary commits itself to locating $\sqrt{n}/100$ input points in each block. In what follows, the adversary will mark some blocks in which it will commit to the internal location of points. The invariant that is kept is that in unmarked blocks, any configuration of input points is still consistent with the answers so far.

At the beginning no block is marked. Now, for each queried rectangle, if the query intersects an unmarked block, then the adversary will answer “not-empty.” In addition it will choose one unmarked block that intersects the given query, mark it, and commit to having all points in that block, in an arbitrary single point in the intersection. If the query intersects only previously marked blocks, then if it contains any of the previous locations in which the adversary has already committed to having input points, then a “non-empty” answer will be given (this is forced). If the query does not include any of the previous locations in which the adversary has committed to having input points, then the adversary will answer “empty.”

Keeping up this way, it is easy to see that the adversary can supply consistent answers to all \sqrt{n} queries.

At the end, since there are $100\sqrt{n}$ blocks while the adversary has marked at most \sqrt{n} blocks, in $99\sqrt{n}$ blocks there is complete freedom as to where the input points are located within such blocks. Now notice that if the adversary chooses to locate all points within a block in one (arbitrary) point, then the MST is of cost $\mathcal{O}(n^{1/4})$, while, if it chooses to locate the points in each unmarked block spread uniformly within the block, then the cost of the tree is $\Omega(\sqrt{n})$. Hence the lower bound follows. \square

Finally, we note that our choice of using $\mathcal{O}(\sqrt{n})$ orthogonal range queries was arbitrary; one can use a different number of queries and obtain a whole range of tradeoffs between the running time and the quality of approximation. As another example in this tradeoff, assume that αn queries are made. The adversary divides the square into βn blocks and proceeds as below. In this situation, the algorithm has no way of telling whether all cells have their points concentrated in one location, resulting in an EMST of at most $\sqrt{\beta n}$, or are distributed in the unmarked blocks as above, resulting in an EMST of at least $\frac{(\beta-\alpha)\sqrt{n}}{\beta}$, giving an approximation ratio of at least $\frac{\beta-\alpha}{\beta\sqrt{\beta}}$. To exploit the tradeoff here, we set $\beta = 1/4$. To obtain an approximation ratio of ε , the number of queries must be at least $n(1/4 - \varepsilon/8)$. One can exploit this tradeoff in other ways also.

4. Two related previous results. We now describe two previous results that we utilize in our EMST algorithms: the concept of Yao graphs [26] and an algorithm for approximating the MST in bounded degree graphs due to Chazelle, Rubinfeld, and Trevisan [10].

4.1. Yao graphs. Yao graphs are Euclidean graphs that relate the EMST to the cone nearest neighbor oracle presented in section 2.1. Fix an integer $d \geq 2$. Let \mathcal{C} be a collection of d -dimensional cones with its apex at the origin such that (a) each cone has an angular diameter² at most θ , where θ is some fixed angle, and (b) $\bigcup_{C \in \mathcal{C}} C = \mathbb{R}^d$. There is always such a collection \mathcal{C} of $\mathcal{O}(d^{3/2} \cdot \sin^{-d}(\theta/2) \cdot \log(d \sin^{-1}(\theta/2)))$ cones (not necessarily disjoint); note that for constant d and θ this bound is $\mathcal{O}(1)$. Yao [26] gives one possible construction for such a collection. For a point $p \in \mathbb{R}^d$ and a cone $C \in \mathcal{C}$, let C_p be $p + C = \{a + p : a \in C\}$, that is, a translation of C so that its apex is at p . Let $N_P\langle p, C \rangle$ be the nearest neighbor of the apex p of C_p in the set $(P \setminus \{p\}) \cap C_p$. Given a point set P and a collection of cones \mathcal{C} , the *Yao graph of P (with respect to \mathcal{C})* is the Euclidean graph G with vertex set P and (undirected) edge set $E = \{(p, q) \mid \exists C \in \mathcal{C} \text{ such that } q = N_P\langle p, C \rangle\}$. That is, each $p \in P$ is connected to its nearest neighbor in each cone that has p at its apex. The following result due to Yao [26] motivates our use of these graphs.

CLAIM 4. [26] *Let P be a point set in \mathbb{R}^d . Let G be the undirected Yao graph for P with $\theta < \pi/3$. Then, the EMST of P is a subgraph of the Yao graph G .*

4.2. Chazelle, Rubinfeld, and Trevisan [10]: Approximate MST in low-degree graphs. Our algorithms make use of a recent algorithm for estimating the weight of MST in graphs due to Chazelle, Rubinfeld, and Trevisan [10]. This algorithm assumes that the input graph (i) is represented by an adjacency list, (ii) has degree at most ν (the full version of [10] allows ν to be the average degree), and (iii) has known minimum and maximum edge weights, where the ratio of the maximum edge weight

²The angular diameter of a cone C in \mathbb{R}^d having its apex at point $p \in \mathbb{R}^d$ is defined as the maximum angle between any two vectors \overrightarrow{px} and \overrightarrow{py} , $x, y \in C$.

to the minimum is Λ . Then, for $0 < \varepsilon < \frac{1}{2}$, the algorithm estimates the weight of the MST with a relative error of at most ε and with probability at least $\frac{3}{4}$, and it runs in time $\mathcal{O}(\nu \cdot \Lambda \cdot \log(\nu \Lambda / \varepsilon) / \varepsilon^3)$. (The authors also give a nearly matching lower bound of $\Omega(\nu \cdot \Lambda / \varepsilon^2)$ on the time complexity of any ε -approximation algorithm for the MST.)

Let $H = (V, E)$ be an input graph having n vertices with maximum degree ν and edge weights in the interval $[1, \Lambda]$. For any $w \in \mathbb{R}$, let $H^{(w)}$ denote the maximal subgraph of H containing edges of weight at most w , and let c_w denote the number of connected components in $H^{(w)}$. The main ingredient of the algorithm from [10] is a procedure called ‘‘approx-number-connected-components,’’ which we call *ACC* from this point on, run on $H^{(w)}$ for estimating c_w for $w = (\frac{1}{2} + i) \cdot \varepsilon$ with $i = 1, 2, \dots, \Lambda/\varepsilon$. For integer weights, the weight of the MST of H is equal to $n - \Lambda + \sum_{j=1}^{\Lambda-1} c_j$. The algorithm uses the above estimations to produce a value which, with probability at least $\frac{3}{4}$, is a $(1 \pm \varepsilon)$ -approximation of the MST of H .

Procedure *ACC* works by sampling $\mathcal{O}(1/\varepsilon^2)$ vertices in H . For each sampled vertex u , a random estimator X_u is computed by traversing $H^{(w)}$ from u (for example, using breadth-first search) with a stochastic stopping rule. X_u is a random variable whose distribution is a function of only the size of the connected component containing u (i.e., the number of vertices reached from u in the traversal) in $H^{(w)}$. The simple relation between these sizes and c_w together with the fact that the distribution of X_u is concentrated around the expected value yields the connection between X_u and c_w . Procedure *ACC* runs in expected time $\mathcal{O}(\nu \varepsilon^{-2} \log(\Lambda/\varepsilon))$. Therefore, the expected running time of the algorithm in [10] is $\mathcal{O}(\Lambda \nu \varepsilon^{-3} \log(\Lambda/\varepsilon))$.

5. A simple estimation for EMST using Yao graphs. The algorithm we present in this section is conceptually an important component of the sublinear algorithm we design later in section 6. It combines the two results described in section 4. Our algorithm uses the cone nearest neighbor oracle and achieves a query complexity of $2^{\mathcal{O}(d)} \cdot \tilde{\mathcal{O}}(\Lambda/\varepsilon^2)$.

Since by Claim 4 the undirected Yao graph G for P contains all edges of the EMST of P , it is natural to try to apply the algorithm of Chazelle, Rubinfeld, and Trevisan to G to estimate the weight of the EMST of P . To do that efficiently, instead of generating G at the beginning of the algorithm, we generate the edges of G (using the cone nearest neighbor queries) only when the edges are needed in the algorithm. That is, whenever the algorithm needs edges adjacent in G to a vertex p , we use the cone nearest neighbor query to obtain the nearest neighbor of p in each cone in $\{p + C\}_{C \in \mathcal{C}}$. Motivated by Claim 4, we set the angular diameter of the cones to $\pi/4$. This creates parts of an *implicit directed* Yao graph \overline{G} on P with edges (p, q) such that there is a $C \in \mathcal{C}$, where $q = N_P(p, C)$.

The above approach has a number of problems. First, the algorithm of Chazelle, Rubinfeld, and Trevisan requires the input graph to be undirected and represented by an adjacency list, whereas in our model, we have fast access to only the *outgoing edges* at a vertex in \overline{G} . Furthermore, the running time is linear in Λ , which can be arbitrarily large. The following lemma helps in overcoming the first difficulty, while the second is tackled in the main algorithm in section 6. The proof of Lemma 1, being a special case of Claim 6, is omitted.

LEMMA 1. *Let n_u^ℓ be the number of vertices in \mathbb{K}_P that are reachable from u using only edges of weight at most ℓ . Let m_u^ℓ be the number of vertices in a directed Yao graph \overline{G} reachable from u using only edges of weight at most ℓ . Then $m_u^\ell = n_u^\ell$.*

Equipped with this lemma, we can modify the algorithm due to Chazelle, Rubinfeld, and Trevisan to obtain its efficient implementation in our model. The only

difference is in procedure *ACC*. We still sample $\mathcal{O}(1/\varepsilon^2)$ vertices and randomly traverse $H^{(w)}$ from the sampled vertices. To implement the traversing algorithm we explore the graph in a breadth-first search fashion by going to the *outgoing* neighbors of the vertices that are closer than the current threshold weight w . Such a procedure can be easily implemented in our model by using the cone nearest neighbor queries; the running time is proportional to the number of the edges traversed. To estimate the value of c_w we use the same estimators as in [10]. Since for each vertex u in the sample, the distribution of X_u depends only on m_u^w , the number of the vertices reachable from u in $H^{(w)}$, by Lemma 1 we can conclude that X_u has the same distribution as in the algorithm of Chazelle, Rubinfeld, and Trevisan [10]. Therefore, the quality of this algorithm of the estimation of EMST of P is the same as in the algorithm of Chazelle, Rubinfeld, and Trevisan [10]. Since the maximum out-degree of the directed Yao graph is $2^{\mathcal{O}(d)}$, the modified procedure *ACC* has complexity identical to that of running the original algorithm of Chazelle, Rubinfeld, and Trevisan in an (undirected) graph with maximum degree $2^{\mathcal{O}(d)}$. Thus, we obtain the following theorem.

THEOREM 1. *Let P be a set of points in \mathbb{R}^d . Assume that the value Λ of the spread of P is known and access to a cone nearest neighbor oracle for P is given. Then, there is an algorithm that outputs a value Υ which, with probability at least $\frac{3}{4}$, approximates the values of EMST(P) to within a factor of $1 \pm \varepsilon$ with query complexity $\tilde{\mathcal{O}}(2^{\mathcal{O}(d)} \cdot \Lambda/\varepsilon^3)$.*

For constant d and ε , this complexity is $\tilde{\mathcal{O}}(\Lambda)$, which is sublinear for $\Lambda = o(n)$. However, on the plane, for example, Λ is known to be $\Omega(\sqrt{n})$, and in general, Λ may be arbitrarily large. In the next section, we discuss our main contribution, which is a truly sublinear-time approximation algorithm whose complexity is independent of Λ .

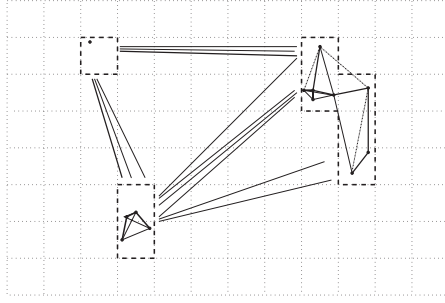
6. Sublinear-time approximation algorithm. In this section we show how the two algorithms from sections 3 and 4 can complement each other. In addition to improving the running time, our algorithm requires a weaker computational model, in which the *cone nearest neighbor query* is replaced by the *cone $(1 + \delta)$ -approximate nearest neighbor query*.

6.1. Overview of the algorithm. In section 6.2, we begin by partitioning a minimal bounding cube \mathcal{BC} of P into blocks of equal size; we then consider only blocks containing points from P . Next, we group blocks that are “close” to each other, calling the resulting clusters *connected block-components*. The algorithm then proceeds in two phases. First, in section 6.5, we show how to approximate the weight of a minimum spanning forest (MSF) of the connected block-components by using the ideas of section 5. Then, in section 6.6, we approximate the optimal way to connect different connected block-components. We prove in Lemma 2 that the MSF of the connected block-components combined with the optimal set of edges joining them approximates the EMST of P .

In our analysis, throughout the entire section we assume that $0 < \varepsilon < \frac{1}{15}$.

6.2. Partitioning the bounding cube. After the translation and scaling of the points in P we can assume that \mathcal{BC} , the bounding cube of P , is $[0, n/\varepsilon]^d$. In particular, the side length is $\mathcal{L} = n/\varepsilon$, and we have a trivial lower bound $\text{EMST}(P) \geq n/\varepsilon$.

We follow the approach from section 3 with small modifications, by extending it to higher dimensions and applying a different stopping procedure. We first partition \mathcal{BC} into 2^d disjoint cubes of equal size, then partition each nonempty cube into 2^d disjoint subcubes, then partition each nonempty subcube further into 2^d subcubes, and so on. Call a block at level i an *active block* if it contains a point from P . Let b_k

FIG. 1. *Block-partitioning and block-components.*

be the number of active blocks at level k (the number of blocks that contain points from P), and let $\Delta_k = \mathcal{L}/2^k$ be the side length of blocks in the k th level of the subdivision. Let $b^* = \max\{\varepsilon^{d/2-3} \sqrt{n}, 2^{d+1}\}$. We stop our subdivision at the first level k_0 such that either $b_{k_0} \geq b^*$ or $\Delta_{k_0} < 2\varepsilon$. Let $b = b_{k_0}$ and $\Delta = \Delta_{k_0}$. Notice that $b \leq 2^d b^*$ and $\Delta \geq \varepsilon$. By our arguments from section 3, the active blocks at level k_0 can be found by querying the range query oracle $\mathcal{O}(b2^d \log(n/(\varepsilon \Delta)))$ times.

6.3. Spanners and connected block-components. For any $t \geq 1$, a t -*spanner* (see, e.g., [7, 14, 17]) for a set S of points in a Euclidean space is any Euclidean graph G with the vertex set S such that for every pair of points $x, y \in S$ there is a path in G between x and y of total length at most $t \cdot |x - y|$.

In our analysis, we will frequently use centers of blocks as the representatives of the blocks. Let B be the set of *centers of active blocks* and let \mathcal{SPN} be a $(1 + \varepsilon/4)$ -spanner of B with $\mathcal{O}(b(4/\varepsilon)^{d-1})$ edges. Such a spanner can be found in time $\mathcal{O}(b \log b + b \log(1/\varepsilon) \varepsilon^{-d}) = \tilde{\mathcal{O}}(\sqrt{n} \varepsilon^{3-d/2})$ [7].

Call two blocks *close* if the distance between their centers in the graph \mathcal{SPN} is at most $\Gamma \cdot \Delta$, where $\Gamma = 14 \sqrt{d}/\varepsilon$. We use equivalence classes of the transitive closure of the relation *close* to define the *connected block-components*. That is, two blocks are in the same connected block-component if there is a sequence of active blocks between them, where every consecutive pair of blocks in the sequence is close. We shall abuse notation and refer also to the partition of P induced by the connected components as *connected block-components*. Notice that all connected block-components can be found in time proportional to the number of edges in \mathcal{SPN} , which is $\mathcal{O}(b(4/\varepsilon)^{d-1})$. See Figure 1 for an example of block-partitioning.

6.4. The EMST of P and connected block-components. We refer to the *spanning forest* of a graph G as a union of spanning trees of the connected components of G . A *minimum spanning forest* of G , denoted by $\text{MSF}(G)$, is a spanning forest of G of minimum weight.

Let E_{in} be the set of edges of \mathbb{K}_P whose endpoints lie within the same connected block-component. Let $W = (\Gamma + \sqrt{d}) \Delta$. We now relate block-components to the distances between points.

Observation 1. Let p and q be an arbitrary pair of points in P .

1. If $|p - q| \leq (\Gamma - 4\sqrt{d}) \Delta$, then p and q are in the same connected block-component.
2. If p and q are in the same connected block-component, then there is a path between p and q consisting of edges in E_{in} that are all of length at most $(\Gamma + \sqrt{d}) \Delta = W$.
3. If $|p - q| > (\Gamma + \sqrt{d}) \Delta = W$ and p and q are in the same connected block-component, then $\text{EMST}(P)$ does not contain the edge pq .

Proof. For any point $p \in P$, we let c_p denote the center of the block at level k_0 that contains p .

To see the first assertion, note that if $|p-q| \leq (\Gamma - 4\sqrt{d})\Delta$, then $|c_p - c_q| \leq |p-q| + \sqrt{d}\Delta \leq (\Gamma - 3\sqrt{d})\Delta$. Therefore, the distance in \mathcal{SPN} between c_p and c_q is at most $(1 + \varepsilon/4)(\Gamma - 3\sqrt{d})\Delta \leq \Gamma\Delta$, which implies the first claim. Next, this also implies the existence of a path $c_p = c^{(0)}, c^{(1)}, \dots, c^{(s)} = c_q$ in \mathcal{SPN} such that $|c^{(i)} - c^{(i+1)}| \leq \Gamma\Delta$ for all i . Clearly, the corresponding path $p = p^{(0)}, p^{(1)}, \dots, p^{(s)} = q$ with $c^{(i)} = c_{p^{(i)}}$ shows the second assertion, since $|p^{(i)} - p^{(i+1)}| \leq |c^{(i)} - c^{(i+1)}| + \sqrt{d}\Delta \leq \Gamma\Delta + \sqrt{d}\Delta$. The third assertion follows from the second and the fact that the (strictly) largest edge in a cycle in a graph cannot be part of its MST. \square

In our algorithm we use the following graphs:

(i) G_{block} is the graph containing all edges in E_{in} of weight at most W . By Observation 1, the connected components of $\text{MSF}(G_{block})$ are identical to the connected block-components, and the MSF of these components is the same as $\text{MSF}(G_{block})$.

(ii) \overline{G}_δ is the *directed* $(1 + \delta)$ -Yao graph that is obtained from \mathbb{K}_P using the cone $(1 + \delta)$ -approximate nearest neighbor oracle. We use the same definitions as in the definition of directed Yao graphs and formally define $N_P^{(1+\delta)}(p, C)$ to be the point that is returned by the cone $(1 + \delta)$ -approximate nearest neighbor oracle for p and C . If $(P \setminus \{p\}) \cap C_p = \emptyset$, then $N_P^{(1+\delta)}(p, C)$ is undefined. Then, \overline{G}_δ is a directed Euclidean graph on P with the edge set containing an edge (p, q) if there is $C \in \mathcal{C}$ such that $q = N_P^{(1+\delta)}(p, C)$.

(iii) \mathcal{M} is the minimum weight subgraph of \mathbb{K}_P that, when added to G_{block} , forms a connected graph.

(iv) G_{out} is the same as \mathbb{K}_P except that the weights of edges in E_{in} are considered to be zero. Observe that the weight of $\text{MST}(G_{out})$ is identical to the weight of \mathcal{M} .

The following lemma displays the two-level nature of the algorithm that we will present.

LEMMA 2. *The sum of the weights of $\text{MSF}(G_{block})$ and $\text{MST}(G_{out})$ is a $(1 + \varepsilon/2)$ -approximation of $\text{EMST}(P)$.*

Proof. We show that the union of $\text{MSF}(G_{block})$ and \mathcal{M} is a spanning tree of \mathbb{K}_P whose weight approximates the weight of $\text{EMST}(P)$ to within a factor of $1 + \varepsilon/2$. From that the lemma follows immediately.

Clearly, the union of $\text{MSF}(G_{block})$ and \mathcal{M} forms a spanning tree of \mathbb{K}_P . To prove the second part of the claim, let us consider an undirected graph G^* obtained from \mathbb{K}_P by decreasing to $(\Gamma - 4\sqrt{d})\Delta$ the weight of every edge in E_{in} having weight larger than $(\Gamma - 4\sqrt{d})\Delta$ and smaller than or equal to W . (Note that we change only the weights of the edges in G_{block} .) Since the weight of every edge decreases by a factor of at most $\frac{W}{(\Gamma - 4\sqrt{d})\Delta} = \frac{(\Gamma + \sqrt{d})\Delta}{(\Gamma - 4\sqrt{d})\Delta} \leq 1 + \varepsilon/2$, we have $\text{MST}(G^*) \geq \text{EMST}(P)/(1 + \varepsilon/2)$. Note further that by Observation 1, each edge in G^* that is not in G_{block} has weight larger than $(\Gamma - 4\sqrt{d})\Delta$. This means that $\text{MST}(G^*)$ must contain an MSF of G_{block} , and hence the weight of the union of $\text{MSF}(G_{block})$ and \mathcal{M} is a $(1 + \varepsilon/2)$ approximation of $\text{EMST}(P)$. \square

6.5. First level—estimating the weight of $\text{MSF}(G_{block})$. In this section we show how to estimate the weight of the MST within a single block-component. This, combined for all block-components, yields an estimate on the weight of $\text{MSF}(G_{block})$. Since our model does not allow constant-time access to the edges of G_{block} , we will use the directed Yao graph \overline{G}_δ to estimate the weight of $\text{MSF}(G_{block})$. Our analysis will explore the relationship between \overline{G}_δ and G_{block} .

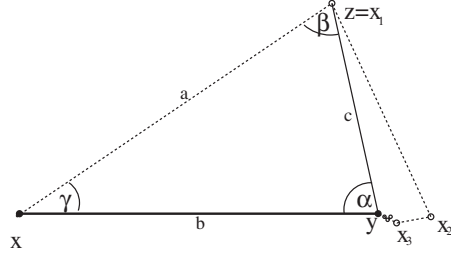


FIG. 2. Illustration for the proof of Claim 6. The figure shows the reachability in $\overline{G_\delta}$. The dashed line is the path showing the connectivity of x and y in $\overline{G_\delta}^{((1+\delta)\tau)}$.

For a weighted graph H denote by $\beta \cdot H$ the graph H with edge weights multiplied by β . Recall that $H^{(r)}$ denotes the subgraph of H consisting of the edges of weight at most r , and c_r is the number of connected components in $G_{block}^{(r)}$. Let n_u^r and m_u^r be the number of vertices in $G_{block}^{(r)}$ and in $\overline{G_\delta}^{(r)}$, respectively, that are reachable from u . Note that $c_r = \sum_{u \in P} 1/n_u^r$. Analogously, define $c_r^* = \sum_{u \in P} 1/m_u^r$. Also, let \hat{c}_r be the number of connected components in $(1 + \delta) \cdot G_{block}^{(r)}$.

The following is from [10].

CLAIM 5. Let $G^{(l)}$ denote the subgraph of a graph G consisting of all the edges of weight at most l . Define $c^{(l)}$ to be the number of connected components in $G^{(l)}$. Then, for integer $w \geq 2$, $M(G) = n - w + \sum_{i=1}^{w-1} c^{(i)}$.

From the above (see also section 4) we have that

$$(1) \quad \text{MSF}(G_{block}^{(r)}) \leq n - rc_r + \sum_{i=1}^{r-1} c_i \leq \text{MSF}(G_{block}^{(r)}) + n.$$

Since we have access only to $\overline{G_\delta}$, we can deal only with the c_r^* 's rather than the c_r 's. To bound the error due to this replacement, now we relate reachability in $\overline{G_\delta}$ to reachability in G_{block} .

CLAIM 6. Let $\varepsilon \leq \frac{1}{5}$ and $\delta \leq \frac{1}{10}$. Then for every r and every $u \in P$, $n_u^{r/(1+\delta)} \leq m_u^r \leq n_u^r$. In particular, $c_{r/(1+\delta)} \geq c_r^* \geq c_r$.

Proof. Let us first note that $m_u^r \leq n_u^r$ follows directly from the definition. To show that $n_u^{r/(1+\delta)} \leq m_u^r$, it suffices to show that for every τ , if a vertex y is reachable in $G_{block}^{(\tau)}$ from a vertex x , then y is reachable from x in $\overline{G_\delta}^{((1+\delta)\tau)}$. Assume that y is reachable from x in $G_{block}^{(\tau)}$; this implies that x and y are in the same connected block-component. Assume further, without loss of generality, that $\tau \leq W$ (indeed, if $\tau > W$, then $G_{block}^{(\tau)} = G_{block}^{(W)}$).

Let z be the $(1 + \delta)$ -approximate nearest neighbor of x (returned by the cone approximate nearest neighbor oracle) in the cone C_x containing y . Clearly, if $z = y$, then the claim holds. So let us assume that $z \neq y$. Let $a = |x - z|$, $b = |x - y|$, $c = |y - z|$, and $\alpha = \angle(xyz)$, $\beta = \angle(xzy)$, and $\gamma = \angle(yxz)$; see Figure 2. Note that since y and z are contained in the cone C_x with the angular diameter $\pi/4$, we have $\gamma \leq \pi/4$.

We first show the following three inequalities: (i) $a \leq (1 + \delta)b$, (ii) $c < b$, and (iii) $\min\{a, c\} \leq b/(1 + \varepsilon)$. Inequality (i) follows directly from the definition of the cone approximate nearest neighbor oracle. To prove inequality (ii), let us suppose that $c \geq b$. Then $\beta \leq \gamma$, and since $\gamma \leq \pi/4$, we obtain that $\alpha \geq \pi/2$. This in

turn implies that $a \geq \sqrt{b^2 + c^2} \geq \sqrt{2}b$, which contradicts the first inequality that $a \leq (1 + \delta)b \leq 1.1 \cdot b$. For inequality (iii), we first use the law of cosines to get $c^2 = a^2 + b^2 - 2ab \cos \gamma \leq a^2 + b^2 - \sqrt{2}ab$, since $\gamma \leq \pi/4$. To show $\min\{a, c\} \leq b/(1 + \varepsilon)$ we assume $a > b/(1 + \varepsilon)$ and show $c \leq b/(1 + \varepsilon)$. Since $a > b/(1 + \varepsilon) \geq \frac{\sqrt{2}}{2}b$, the expression $a^2 + b^2 - \sqrt{2}ab$ increases with a . Therefore, by inequality (i) we obtain

$$\begin{aligned} c^2 &\leq a^2 + b^2 - \sqrt{2}ab \leq ((1 + \delta)b)^2 + b^2 - \sqrt{2}(1 + \delta)b^2 \\ &= b^2((2 - \sqrt{2})(1 + \delta) + \delta^2) \leq (b/(1 + \varepsilon))^2, \end{aligned}$$

where the last inequality holds for $\varepsilon \leq \frac{1}{5}$ and $\delta \leq \frac{1}{10}$.

Now, we prove the claim using inequalities (i)–(iii). Assume, without loss of generality, that $|x - y| \leq \tau$; otherwise apply the following arguments to all edges on the path between x and y in $G_{block}^{(\tau)}$ (all the edges on this path are of length at most τ). We define inductively the sequence $x = x_0, x_1, x_2, \dots, y$ such that for every i , if $x_i \neq y$, then x_{i+1} is the $(1 + \delta)$ -approximate nearest neighbor of x_i in the cone C_{x_i} containing y . By inequality (ii), the sequence $|x_i - y|$ is strictly decreasing. This immediately implies that $x_i = y$ for some i , and so the sequence is finite.

Next, we show inductively that each x_i is in the same connected block-component as y . Suppose that x_i is in the same connected block-component as y . Since the sequence $|x_i - y|$ is decreasing and since $|x - y| \leq \tau \leq W = (\Gamma + \sqrt{d}) \cdot \Delta$, we obtain

$$\frac{|x_i - y|}{1 + \varepsilon} \leq \frac{|x - y|}{1 + \varepsilon} < \frac{(\Gamma + \sqrt{d}) \cdot \Delta}{1 + \varepsilon} \leq \frac{(\Gamma + \sqrt{d}) \cdot \Delta}{1 + \varepsilon}.$$

Therefore, using inequality (iii) with $x = x_i$ and $z = x_{i+1}$, we obtain

$$\min\{|x_i - x_{i+1}|, |x_{i+1} - y|\} \leq \frac{|x_i - y|}{1 + \varepsilon} \leq (\Gamma - \sqrt{d}) \cdot \Delta.$$

Hence, by Observation 1, either x_i and x_{i+1} are in the same connected block-component or x_{i+1} and y are in the same connected block-component. In either case, the transitivity ensures that x_{i+1} and y are in the same connected block-component. We finally observe that inequality (i) implies that $|x_i - x_{i+1}| \leq (1 + \delta)|x_i - y|$, and since $|x_i - y| \leq |x - y|$, we obtain $|x_i - x_{i+1}| \leq (1 + \delta)|x - y|$. Hence, the sequence $x = x_0, x_1, x_2, \dots, y$ corresponds to a path contained in a connected block-component having all edges of length at most $(1 + \delta)\tau$. This implies that y is reachable from x in $\overline{G}_\delta^{((1 + \delta)\tau)}$. \square

Let $W' = \lceil W(1 + \delta) \rceil$. Motivated by inequality (1), let us introduce an estimator \mathcal{A} for the value of $\text{MSF}(G_{block})$:

$$\mathcal{A} = n + \sum_{i=1}^{W'-1} c_i^* - W' \cdot c_{W'}^*.$$

We now analyze the quality of this estimator.

LEMMA 3. $\text{MSF}(G_{block}) \leq \mathcal{A} \leq (1 + \delta) \cdot \text{MSF}(G_{block}) + n$.

Proof. Let us first recall that $\hat{c}_r = c_{r/(1 + \delta)}$. Next, let us observe that if $r \geq W$, then $c_r = c_W$. As a corollary, $c_{W'}^* = c_W = c_{W'}$. With this, we have the following

sequence of inequalities:

$$\begin{aligned}
\text{MSF}(G_{block}) &\leq n + \sum_{i=1}^{W-1} c_i - W \cdot c_W \leq n + \sum_{i=1}^{W'-1} c_i - W' \cdot c_{W'} \\
&= n + \sum_{i=1}^{W'-1} c_i - W' \cdot c_{W'}^* \leq n + \sum_{i=1}^{W'-1} c_i^* - W' \cdot c_{W'}^* \\
&= \mathcal{A} \leq n + \sum_{i=1}^{W'-1} c_{i/(1+\delta)} - W' \cdot c_W = n + \sum_{i=1}^{W'-1} \hat{c}_i - W' \cdot \hat{c}_{W'} \\
&\leq \text{MSF}((1+\delta) \cdot G_{block}) + n = (1+\delta) \cdot \text{MSF}(G_{block}) + n.
\end{aligned}$$

The first inequality is due to inequality (1). The second one follows from the observation mentioned at the beginning of this proof, as does the next one, combined with Claim 6. The last inequality is implied by inequality (1). \square

We now modify the algorithm of Chazelle, Rubinfeld, and Trevisan [10] to obtain a good approximation of \mathcal{A} . Let us first note that as in section 5, we can easily traverse the graph $\overline{G}_\delta^{(r)}$: each time we want to access all edges incident to a point $p \in P$, we first ask the cone approximate nearest neighbor queries to all cones C_p and then for each nearest neighbor q of p in C_p , we verify whether $|p - q| \leq r$ and whether the blocks to which p and q belong are in the same connected block-component. The first test is a simple $\mathcal{O}(1)$ -time calculation, while the second requires the computation of the connected block-components. Establishing this, we can apply the approach from sections 4.2 and 5 to estimate the value $c^* = \sum_{r=1}^{W'-1} c_r^*$, and hence to estimate the value of \mathcal{A} . For this, we run procedure *ACC* to get an estimator X_r to c_r^* for all $r = 1, 2, \dots, W'$, and we now show that $X = \sum_{r=1}^{W'-1} X_r$ is a good approximation to $c^* = \sum_{r=1}^{W'-1} c_r^*$.

We have from [10] that after using procedure *ACC*(G, ε, W), where G denotes the input graph, ε the approximation parameter, and W the stopping condition for the breadth-first search, we have $c^* - n/W \leq \text{EX} \leq c^*$ and $\text{var}X \leq 2n c^*/s$, where s is the number of initial random choices for vertices. Our algorithm yields similar properties; thus, we have the following:

$$c^* - n/2 \leq \text{EX} \leq c^*$$

and

$$\text{var}X \leq 2n c^*/s,$$

where s is the number of random choices of initial vertices in *ACC*. Next, using the bounds above, the fact that $\text{EMST}(P) \geq n/\varepsilon$, and Chebyshev's inequality, we have

$$\begin{aligned}
\Pr[|X - c^*| \geq \varepsilon/2 \cdot \text{EMST}(P)] &\leq \Pr[|X - \text{EX}| \geq \varepsilon/4 \cdot \text{EMST}(P)] \\
&\leq \frac{16 \text{var}X}{\varepsilon^2 \cdot (\text{EMST}(P))^2} \leq \frac{32 \cdot n \cdot c^*}{\varepsilon^2 \cdot s \cdot (\text{EMST}(P))^2}.
\end{aligned}$$

We argue that $32n c^*/(\varepsilon^2 s (\text{EMST}(P))^2) = \mathcal{O}(\frac{1}{\varepsilon s})$ or, alternatively, that $(\text{EMST}(P))^2 = \Omega(nc^*/\varepsilon)$. Indeed, if $c^* \leq 2n/\varepsilon$, then $(\text{EMST}(P))^2 \geq (n/\varepsilon)^2 \geq 2n c^*/\varepsilon$ by our assumption in section 6.2. Otherwise, we have to use a stronger lower bound for $\text{EMST}(P)$.

By Lemma 2, we have

$$\text{EMST}(P) = \Omega(\text{MSF}(G_{\text{block}}) + \text{MST}(G_{\text{out}})) = \Omega(\text{MSF}(G_{\text{block}}) + W' \cdot (c'_W - 1)).$$

Next, by Lemma 3, we have

$$(1 + \delta) \cdot \text{MSF}(G_{\text{block}}) \geq \mathcal{A} - n = c^* - W' \cdot c_{W'}^*.$$

$$\text{EMST}(P) = \Omega(c^* - W' \cdot c_{W'}^* + W' \cdot (c'_W - 1)) = \Omega(c^*),$$

from which it follows that for $c^* > 2n/\varepsilon$ we have $(\text{EMST}(P))^2 = (\Omega(c^*))^2 = \Omega(nc^*/\varepsilon)$, as required.

Summarizing the discussion above, we always have $(\text{EMST}(P))^2 = \Omega(nc^*/\varepsilon)$ and hence

$$\Pr[|X - c^*| \geq \varepsilon/2 \cdot \text{EMST}(P)] \leq \mathcal{O}(\frac{1}{\varepsilon s}).$$

Therefore, if we choose $s = \mathcal{O}(1/\varepsilon)$, then we obtain

$$\Pr[|X - c^*| \geq \varepsilon/2 \cdot \text{EMST}(P)] \leq 1/4.$$

Next, observe that $c_{W'}^*$ is nothing but the number of connected block-components, which is known to the algorithm that computes the connected block-components. This leads to an efficient algorithm that calculates $\mathcal{A}' = n + X - W' \cdot c_{W'}^*$, for which $\Pr[|\mathcal{A}' - \mathcal{A}| > \frac{1}{2} \varepsilon \cdot \text{EMST}(P)] \leq 1/4$. The complexity of this algorithm follows from the analysis in section 5 and from [10] (result stated in section 4.2) and is $\tilde{\mathcal{O}}(W \cdot 2^{\mathcal{O}(d)}/\varepsilon)$ cone approximate nearest neighbor queries. The algorithm approximates c^* to within an additive error of n with probability at least $\frac{3}{4}$ (see [10], presented here in section 4.2) and hence approximates $\text{EMST}(P)$ to within an additive error of $\frac{1}{2} \varepsilon \cdot \text{EMST}(P) + \delta \cdot \text{EMST}(P) + n = (\delta + \frac{1}{2} \varepsilon) \cdot \text{EMST}(P) + n$.

We note that by scaling down all weights by a factor $\lambda > 1$, applying the algorithm above, and then rescaling to the original weight, we decrease the running time by a factor of λ and increase the additive error by the same factor. In this way we obtain an algorithm that performs $\tilde{\mathcal{O}}(W \cdot 2^{\mathcal{O}(d)}/(\lambda \varepsilon))$ cone approximate nearest neighbor queries and we achieve an additive error of $(\delta + \frac{1}{2} \varepsilon) \cdot \text{EMST}(P) + \lambda \cdot n$.

Let us examine the term W/λ in the running time and the additive error term $(\delta + \frac{1}{2} \varepsilon) \cdot \text{EMST}(P) + \lambda \cdot n$. Recall that there are two possible termination states: $b \geq b^*$ or $\Delta < 2\varepsilon$.

Consider first the case $b \geq b^*$. Since P has b active blocks of size Δ we have that $\text{EMST}(P) \geq \frac{1}{2} \Delta (\lceil b/2^d \rceil - 1)$. This bound is achieved by considering a subdivision of the active block to $2^d b$ subcubes of size $\Delta/2$. Now color these subblocks with 2^d different colors, using the same arrangement of colors for each of the original active blocks. This induces a partition of the active blocks into 2^d monochromatic sets. There has to be a set of $\lceil b/2^d \rceil$ points in P from different active blocks that are colored the same. Clearly, the minimal distance between these points must be at least $\Delta/2$, and hence the bound. Once we established that $\text{EMST}(P) \geq \frac{1}{2} \Delta (\lceil b/2^d \rceil - 1)$, we use the inequalities $b \geq b^* \geq 2^{d+1}$ to get $\text{EMST}(P) \geq \frac{1}{4} \cdot \Delta \cdot b/2^d$. Setting $\lambda = \frac{\Delta b \varepsilon}{8 \cdot 2^d n}$ we upper bound the relative error by

$$\delta + \varepsilon/2 + \frac{\lambda \cdot n}{\frac{1}{4} \cdot \Delta \cdot b/2^d} = 1 + \delta + \varepsilon.$$

The running time, using the fact that $b \geq b^* \geq \varepsilon^{d/2-3} \sqrt{n}$, is bounded by

$$\tilde{\mathcal{O}}(W'/(\lambda\varepsilon)) = \tilde{\mathcal{O}}(\sqrt{d} \cdot 2^d \cdot n/(b\varepsilon^3)) \leq \tilde{\mathcal{O}}(\sqrt{n} \cdot 2^d \cdot \sqrt{d}/\varepsilon^{d/2}) = \tilde{\mathcal{O}}(\sqrt{n}/\varepsilon^{d/2}).$$

On the other hand, when $\Delta < 2\varepsilon$, we use the trivial lower bound $\text{EMST}(P) \geq n/\varepsilon$ and by setting $\lambda = 1/2$ we obtain a multiplicative error of $1 + \delta + \varepsilon$. In this case note that $W' = \lceil (1 + \delta) \cdot \Delta \cdot \sqrt{d} \cdot (1 + 14/\varepsilon) \rceil = \mathcal{O}(1)$. And so we bound the running time by

$$W/(\lambda\varepsilon^3) = \tilde{\mathcal{O}}(\varepsilon^{-3}) \leq \tilde{\mathcal{O}}(\sqrt{n}/\varepsilon^{2+d/2})$$

for $d \geq 2$. Thus we have the following lemma.

LEMMA 4. *Given the graph G_{block} , there is an algorithm that estimates with probability at least $\frac{3}{4}$ the weight of $\text{MSF}(G_{\text{block}})$ to within a multiplicative relative error of $\delta + \varepsilon$. The algorithm requires $\tilde{\mathcal{O}}(\sqrt{n}/\varepsilon^{2+d/2})$ range queries and cone $(1 + \delta)$ -approximate nearest neighbor queries (for $\delta \leq \varepsilon/6$).*

6.6. Second level—estimating the weight of $\text{MST}(G_{\text{out}})$. Let Q be the complete undirected graph with the vertex set B , the set of active blocks, and with the edge weights equal to the Euclidean distances between the corresponding block-centers if the blocks are in different connected block-components, and zero otherwise. Arguments similar in spirit to those used in Observation 1 can be used to show the following lemma.

LEMMA 5. $(1 - \varepsilon/2) \cdot \text{EMST}(Q) \leq \text{MST}(G_{\text{out}}) \leq (1 + \varepsilon/2) \cdot \text{EMST}(Q)$.

Proof. Given $\text{EMST}(Q)$, consider a subgraph \hat{G} of G_{out} defined as follows. Include in \hat{G} the MST of the subgraph in each connected block-component of G_{out} ; notice that the total weight of the MST is 0. Then, for each nonzero weight edge e in $\text{EMST}(Q)$ connecting two connected block-components, include in \hat{G} the lowest weight edge e' in G_{out} that connects the two block-components joined by e . Due to the construction of Q , the weight of e is at least $\frac{\Gamma \cdot \Delta}{1 + \varepsilon/4}$ and the weight of e' exceeds that of e by at most $\sqrt{d} \Delta$, which is less than an $\varepsilon/2$ fraction of the weight of e . Also note that any two connected block-components in Q will be connected by at most one edge in $\text{EMST}(Q)$; thus such an e' will always be found. It is easy now to see that \hat{G} is a spanning tree of $\text{MST}(G_{\text{out}})$ of weight at most $\text{EMST}(Q)(1 + \varepsilon/2)$. Thus, $\text{MST}(G_{\text{out}}) \leq (1 + \varepsilon/2) \cdot \text{EMST}(Q)$.

To see the lower bound, we follow similar arguments. We start with $\text{MST}(G_{\text{out}})$ and construct a spanning tree of Q in a similar fashion—with the exception that for an edge in $\text{MST}(G_{\text{out}})$ joining two block-components, we include an edge from Q which joins these components by linking two block-centers—again with an increase in weight of at most an $\varepsilon/2$ fraction. Using a similar argument as above, we can show that $(1 - \varepsilon/2) \cdot \text{EMST}(Q) \leq \text{MST}(G_{\text{out}})$, which concludes the proof. \square

In view of Lemma 5, to obtain a good estimation of the weight of $\text{MST}(G_{\text{out}})$ it is sufficient to estimate the weight of an MST of Q .

We could find an MST of Q by calling any algorithm that finds an MST in graphs. However, any such algorithm requires time significantly more than $\mathcal{O}(b)$ time (at least for $d \geq 3$), because Q contains $\Theta(b^2)$ edges. To improve the running time to $\tilde{\mathcal{O}}(b\varepsilon^{1-d}) = \tilde{\mathcal{O}}(\sqrt{n}/\varepsilon^{2+d/2})$ we use \mathcal{SPN} , which is the $(1 + \varepsilon/4)$ -spanner of B (having $\mathcal{O}(b(1/\varepsilon)^{d-1})$ edges) defined in section 6.2. Let \mathcal{F} be any spanning forest of the subgraph of Q induced by the edges of weight 0. It is easy to see that the weight of any MST of Q is identical to the weight of an MST of Q that uses the edges from \mathcal{F} .

We create a new graph SG with the vertex set B and the edge set which is the union of the edges in \mathcal{F} and the spanner edges. Note that the number of edges

in \mathcal{F} is less than the number of spanner edges. Then we apply, for instance, the classical Kruskal's algorithm to find in time $\mathcal{O}(b \varepsilon^{1-d} \log(b/\varepsilon^{d-1})) = \tilde{\mathcal{O}}(\sqrt{n}/\varepsilon^{2+d/2})$ a minimum weight spanning tree of SG . It is easy to see that the obtained spanning tree of B is a spanning tree of B that uses edges from \mathcal{F} , since those edges are of weight 0 and do not form a cycle, and thus are bound to be picked by Kruskal's algorithm. Also, note that the resulting spanning tree's weight is at most $(1 + \varepsilon/4)$ times the minimum, due to the properties of a spanner. This allows us to summarize the discussion in this section in the following lemma.

LEMMA 6. *There is an algorithm which, given as input the graph G_{block} , estimates the weight of \mathcal{M} to within a relative error of $\frac{3}{4}\varepsilon$ with running time $\tilde{\mathcal{O}}(\sqrt{n}/\varepsilon^{2+d/2})$.*

Our analysis in this section can be slightly improved in the case where $d = 2$. In this case, one can simplify the arguments to achieve the running time of $\mathcal{O}(b \log b) = \mathcal{O}(\sqrt{n} \varepsilon^{-2} \log(\sqrt{n}/\varepsilon^2))$. The main difference is that for $d = 2$ we can find in $\mathcal{O}(b \log b)$ time an MST of Q using a modification of the classical algorithm due to Shamos and Hoey [22] for finding an EMST in \mathbb{R}^2 . We first construct (in $\mathcal{O}(b \log b)$ time) the Delaunay triangulation of B . Next, we observe that there is an MST of Q that uses only edges of cost zero and edges from the Delaunay triangulation (for example, this follows from the proof of Lemma 1 in [14] or from [22]). Therefore, an MST of Q can be found by calling, for example, Kruskal's algorithm on the subgraph of Q containing only the edges from the Delaunay triangulation of B and the edges of the forest \mathcal{F} as defined above. Since the number of edges in the Delaunay triangulation and in \mathcal{F} is $\mathcal{O}(b)$, this algorithm finds $\text{MST}(Q)$ in $\mathcal{O}(b \log b) = \tilde{\mathcal{O}}(\sqrt{n}/\varepsilon^2)$ time.

6.7. Estimating the weight of $\text{MSF}(G_{block}) \cup \text{MST}(G_{out})$. We can now summarize our algorithm for estimating the EMST of any set of points in \mathbb{R}^d . We use the fact that $\mathcal{L} = \Theta(n/\varepsilon)$ and apply Lemmas 2, 4, and 6 to estimate the weight of the EMST. Summing up the error terms in our estimation stated in each of those lemmas, we get that the multiplicative relative error is at most $\delta + 2\frac{1}{4}\varepsilon$ with probability at least $\frac{3}{4}$. Using $\varepsilon' = \varepsilon/3$ as the input parameter for our algorithm, we can conclude with the following main theorem of the paper.

THEOREM 2. *Let P be a set of n points in \mathbb{R}^d for a constant d . Let ε be any real number, $0 < \varepsilon < \frac{1}{15}$, and let $\delta \leq \varepsilon/4$. There is an algorithm that with probability at least $\frac{3}{4}$ estimates the weight of a Euclidean minimum spanning tree of P with a relative error of at most ε . This algorithm runs in $\tilde{\mathcal{O}}(\sqrt{n}/\varepsilon^{2+d/2})$ time and requires $\tilde{\mathcal{O}}(\sqrt{n}/\varepsilon^{2+d/2})$ orthogonal range queries, $\tilde{\mathcal{O}}(\sqrt{n}/\varepsilon^{2+d/2})$ cone $(1 + \delta)$ -approximate nearest neighbor queries, and a single minimal bounding cube of P .*

Let us also mention that the remark at the end of section 6.6 can be incorporated here to improve the complexity bounds in the most basic case when $d = 2$, that is, for the EMST problem on the Euclidean plane. Then, we obtain the following theorem.

THEOREM 3. *Let P be a set of n points in \mathbb{R}^2 . Let ε be any real number, $0 < \varepsilon < \frac{1}{15}$, and let $\delta \leq \varepsilon/4$. There is an algorithm that, with probability at least $\frac{3}{4}$, estimates the weight of a Euclidean minimum spanning tree of P with a relative error of at most ε . This algorithm runs in $\tilde{\mathcal{O}}(\sqrt{n}/\varepsilon^2)$ time and requires $\tilde{\mathcal{O}}(\sqrt{n}/\varepsilon^2)$ orthogonal range queries, $\tilde{\mathcal{O}}(\sqrt{n}/\varepsilon^2)$ cone $(1+\delta)$ -approximate nearest neighbor queries, and a single minimal bounding cube of P .*

Acknowledgments. We thank Bernard Chazelle and Sarel Har-Peled for helpful discussions on geometric data structures. We also thank the anonymous referees for their helpful comments.

REFERENCES

- [1] P. K. AGARWAL, *Range searching*, in Handbook of Discrete and Computational Geometry, CRC Press Ser. Discrete Math. Appl., CRC Press, Boca Raton, FL, 1997, pp. 575–598.
- [2] P. K. AGARWAL, H. EDELSBRUNNER, O. SCHWARZKOPF, AND E. WELZL, *Euclidean minimum spanning trees and bichromatic closest pairs*, Discrete Comput. Geom., 6 (1991), pp. 407–422.
- [3] P. K. AGARWAL AND J. ERICKSON, *Geometric range searching and its relatives*, in Advances in Discrete and Computational Geometry, Contemp. Math. 223, AMS, Providence, RI, 1999, pp. 1–56.
- [4] S. ARYA, D. M. MOUNT, AND M. SMID, *Dynamic algorithms for geometric spanners of small diameter: Randomized solutions*, Discrete Comput. Geom., 13 (1999), pp. 91–107.
- [5] S. ARYA AND M. SMID, *Efficient construction of a bounded-degree spanner with low weight*, Algorithmica, 17 (1997), pp. 33–54.
- [6] M. DE BERG, M. VAN KREVELD, M. OVERMARS, AND O. SCHWARZKOPF, *Computational Geometry. Algorithms and Applications*, Springer-Verlag, Berlin, 1997.
- [7] P. B. CALLAHAN AND S. R. KOSARAJU, *Faster algorithms for some geometric graph problems in higher dimensions*, in Proceedings of the 4th Annual ACM-SIAM Symposium on Discrete Algorithms, ACM, New York, SIAM, Philadelphia, 1993, pp. 291–300.
- [8] B. CHAZELLE, *Filtering search: A new approach to query-answering*, SIAM J. Comput., 15 (1986), pp. 703–724.
- [9] B. CHAZELLE, *A functional approach to data structures and its use in multidimensional searching*, SIAM J. Comput., 17 (1988), pp. 427–462.
- [10] B. CHAZELLE, R. RUBINFELD, AND L. TREVISAN, *Approximating the minimum spanning tree weight in sublinear time*, Automata, Languages and Programming, Lecture Notes in Comput. Sci. 2076, Springer-Verlag, Berlin, 2001, pp. 190–200.
- [11] A. CZUMAJ AND C. SOHLER, *Property testing with geometric queries*, in Algorithms—ESA—2001 (Århus), Lecture Notes in Comput. Sci. 2161, Springer-Verlag, Berlin, 2001, pp. 266–277.
- [12] A. CZUMAJ AND C. SOHLER, *Estimating the weight of metric minimum spanning trees in sublinear-time*, in Proceedings of the 36th Annual ACM Symposium on Theory of Computing (STOC), 2004, pp. 175–183.
- [13] D. Z. DU AND F. K. HWANG, *Gilbert–Pollack conjecture on Steiner ratio is true*, Algorithmica, 7 (1992), pp. 121–135.
- [14] D. EPPSTEIN, *Spanning trees and spanners*, in Handbook of Computational Geometry, North-Holland, Amsterdam, 1997, pp. 425–461.
- [15] D. EPPSTEIN, *Dynamic Euclidean minimum spanning trees and extrema of binary functions*, Discrete Comput. Geom., 13 (1995), pp. 111–122.
- [16] J. ERICKSON, *On the relative complexity of some geometric problems*, in Proceedings of the 7th Canadian Conference on Computational Geometry (CCCG), Quebec City, 1995, pp. 85–90.
- [17] J. GUDMUNDSSON, C. LEVCOPOULOS, AND G. NARASIMHAN, *Fast greedy algorithms for constructing sparse geometric spanners*, SIAM J. Comput., 31 (2002), pp. 1479–1500.
- [18] P. INDYK, *Algorithms for dynamic geometric problems over data streams*, in Proceedings of the 36th Annual ACM Symposium on Theory of Computing (STOC), 2004, pp. 373–380.
- [19] R. M. KARP AND J. M. STEELE, *Probabilistic analysis of heuristics*, The Traveling Salesman Problem, Wiley-Intersci. Ser. Discrete Math., E. L. Lawler, J. K. Lenstra, A. H. G. Rinnooy Kan, and D. B. Shmoys, eds., Wiley, Chichester, UK, 1985, pp. 181–205.
- [20] G. S. LUEKER, *A data structure for orthogonal range queries*, in Proceedings of the 19th IEEE Symposium on Foundations of Computer Science (FOCS), 1978, pp. 28–34.
- [21] J. RUPPERT AND R. SEIDEL, *Approximating the d -dimensional complete Euclidean graph*, in Proceedings of the 3rd Canadian Conference on Computational Geometry (CCCG), Vancouver, 1991, pp. 207–210.
- [22] M. I. SHAMOS AND D. HOEY, *Closest-point problems*, in Proceedings of the 16th IEEE Symposium on Foundations of Computer Science (FOCS), pp. 151–162, 1975.
- [23] C. SOHLER, *Property Testing and Geometry*, Ph.D. dissertation, HNI-Verlagschriftenreihe, Vol. 119, Heinz-Nixdorf Institut und University of Paderborn, 2003; available from <http://www.upb.de/fachbereich/AG/agmadh/WWW/english/csohler/Diss.ps.gz>.
- [24] V. V. VAZIRANI, *Approximation Algorithms*, Springer-Verlag, Berlin, 2001.
- [25] D. E. WILLARD, *Predicate-Oriented Database Search Algorithms*, Ph.D. thesis (report TR-20-78), Harvard University, Aiken Computation Lab, Cambridge, MA, 1978.
- [26] A. C.-C. YAO, *On constructing minimum spanning trees in k -dimensional spaces and related problems*, SIAM J. Comput., 11 (1982), pp. 721–736.

POLYNOMIAL TIME APPROXIMATION SCHEMES FOR MAX-BISECTION ON PLANAR AND GEOMETRIC GRAPHS*

KLAUS JANSEN[†], MAREK KARPINSKI[‡], ANDRZEJ LINGAS[§], AND EIKE SEIDEL[†]

Abstract. The max-bisection and min-bisection problems are to find a partition of the vertices of a graph into two equal size subsets that, respectively, maximizes or minimizes the number of edges with endpoints in both subsets.

We design the first polynomial time approximation scheme for the max-bisection problem on arbitrary planar graphs solving a long-standing open problem. The method of solution involves designing exact polynomial time algorithms for computing optimal partitions of bounded treewidth graphs, in particular max- and min-bisection, which could be of independent interest.

Using a similar method we design also the first polynomial time approximation scheme for max-bisection on unit disk graphs (which could also be easily extended to other geometrically defined graphs).

Key words. combinatorial optimization, NP-hardness, approximation algorithms, polynomial time approximation schemes, graph bisection, planar graphs

AMS subject classifications. 90C27, 68W25, 68W40, 68Q17, 68Q25, 68R10, 05C62

DOI. 10.1137/S009753970139567X

1. Introduction. The max-bisection and min-bisection problems, i.e., the problems of constructing a halving of the vertex set of a graph that, respectively, maximizes or minimizes the number of edges across the partition, belong to the basic combinatorial optimization problems.

The best-known approximation algorithm for max-bisection yields a solution whose size is at least 0.701 times the optimum [16] (cf. [12, 14, 23]), whereas the best-known approximation algorithm for min-bisection achieves “solely” a log-square approximation factor [11]. The former factor for max-bisection is considerably improved for regular graphs to 0.795 in [10], whereas the latter factor for min-bisection is improved for graphs excluding any fixed minor (e.g., planar graphs) to a logarithmic one in [11]. For dense graphs, Arora, Karger, and Karpinski give polynomial time approximation schemes for max- and min-bisection in [2].

In this paper, we study the max-bisection and min-bisection problems on bounded treewidth graphs and on planar graphs. Both graph families are known to admit exact polynomial time algorithms for max-cut, i.e., for finding a bipartition that maximizes the number of edges with endpoints in both sets in the partition [9, 15].

Our first main result are exact polynomial time algorithms for finding a partition of a bounded treewidth graph into two sets of a priori given cardinalities, respectively maximizing or minimizing the number of edges with endpoints in both sets. Thus, in

*Received by the editors September 28, 2001; accepted for publication (in revised form) March 24, 2005; published electronically September 8, 2005. An extended abstract of a preliminary version of this paper appeared in *Proceedings of the 18th Annual Symposium on Theoretical Aspects of Computer Science*, Lecture Notes in Comput. Sci. 2010, Springer, Berlin, pp. 365–375.

<http://www.siam.org/journals/sicomp/35-1/39567.html>

[†]Institut für Informatik und Praktische Mathematik, Christian-Albrechts-Universität zu Kiel, 24098 Kiel, Germany (kj@informatik.uni-kiel.de, ese@informatik.uni-kiel.de).

[‡]Department of Computer Science, University of Bonn, 53117 Bonn, Germany (marek@cs.uni-bonn.de).

[§]Department of Computer Science, Lund University, 22100 Lund, Sweden (Andrzej.Lingas@cs.lth.se).

particular, we obtain polynomial time algorithms for max-bisection and min-bisection on bounded treewidth graphs.

The complexity and approximability status of max-bisection on planar graphs have been long-standing open problems. Contrary to the status of planar max-cut, planar max-bisection has been proven only recently to be NP-hard in exact setting by Jerrum [18]. (For the sake of completeness of our paper, we provide this proof with his agreement in section 3.) The technique used in his proof is similar to the method used by Barahona [4] to prove NP-hardness of the planar spin glass problem within a magnetic field. Karpinski, Kowaluk, and Lingas observed in [19] that the max-bisection problem for planar graphs does not fall directly into the Khanna–Motwani syntactic framework for planar optimization problems [20]. On the other hand, they provided a polynomial time approximation scheme (PTAS) for max-bisection in planar graphs of sublinear maximum degree. (In fact, their method implies that the size of max-bisection is very close to that of max-cut in planar graphs of sublinear maximum degree.)

Our second main result is the first PTAS for the max-bisection problem for arbitrary planar graphs. It is obtained by combining (via tree-typed dynamic programming) the original Baker’s method of dividing the input planar graph into families of k -outerplanar graphs [3] with our method of finding maximum partitions of bounded treewidth graphs.

Note that the NP-hardness of exact planar max-bisection makes our PTAS result best possible under usual assumptions.

Interestingly, our PTAS for planar max-bisection can be easily modified to a PTAS for the problem of min-bisection on planar graphs in the very special case where the min-bisection is relatively large, i.e., cuts $\Omega(n \log \log n / \log n)$ edges.

Unit disk graphs are another important class of graphs defined by the geometric conditions on a plane. An undirected graph is a unit disk graph if its vertices can be put in one-to-one correspondence with disks of equal radius in the plane in such a way that two vertices are joined by an edge if and only if the corresponding disks intersect. Tangent disks are considered to intersect.

Our third main result is the first PTAS for the max-bisection problem on unit disk graphs. The scheme can be easily generalized to include other geometric intersection graphs. It is obtained by combining (again via tree-typed dynamic programming) the idea of Hunt et al. of dividing the input graph defined by plane conditions into families of subgraphs [17] with the aforementioned known methods of finding maximum partitions of dense graphs [2].

The structure of our paper is as follows. The next section complements the introduction with basic definitions and facts. In section 3, we provide the proof of NP-hardness of planar max-bisection communicated by Jerrum [18]. In section 4, the algorithms for optimal partitions of bounded treewidth graphs are given. Section 5 presents the PTAS for planar max-bisections. In section 6, we make several observations on the approximability of planar min-bisection. Finally, section 7 describes the PTAS for max-bisection on unit disk graphs. In conclusion we notice that the same technique can be applied also for other geometric intersection graphs.

2. Preliminaries. We start with formulating the underlying optimal graph partition problems.

DEFINITION 2.1. *A partition of a set of vertices of an undirected graph G into two sets X, Y is called an $(|X|, |Y|)$ -partition of G . The edges of G with one endpoint in X and the other in Y are said to be cut by the partition. The size of an (l, k) -partition*

is the number of edges which are cut by it. An (l, k) -partition of G is said to be a maximum (l, k) -partition of G if it has the largest size among all (l, k) -partitions of G . An (l, k) -partition of G is a bisection if $l = k$. A bisection of G is a max-bisection or a min-bisection of G if it, respectively, maximizes or minimizes the number of cut edges. An (l, k) -partition of G is a max-cut of G if it has the largest size among all (l', k') -partitions of G . The max-cut problem is to find a max-cut of a graph. Analogously, the max-bisection problem is to find a max-bisection of a graph. The min-cut problem and the min-bisection problem are defined analogously.

The notion of treewidth of a graph was originally introduced by Robertson and Seymour [22]. It has turned out to be equivalent to several other interesting graph theoretic notions, e.g., the notion of partial k -trees [1, 5].

DEFINITION 2.2. A tree-decomposition of a graph $G = (V, E)$ is a pair $(\{X_i \mid i \in I\}, T = (I, F))$, where $\{X_i \mid i \in I\}$ is a collection of subsets of V , and $T = (I, F)$ is a tree, such that the following conditions hold:

1. $\bigcup_{i \in I} X_i = V$.
2. For all edges $(v, w) \in E$, there exists a node $i \in I$, with $v, w \in X_i$.
3. For every vertex $v \in V$, the subgraph of T , induced by the nodes $\{i \in I \mid v \in X_i\}$, is connected.

The treewidth of a tree-decomposition $(\{X_i \mid i \in I\}, T = (I, F))$ is $\max_{i \in I} |X_i| - 1$. The treewidth of a graph is the minimum treewidth over all possible tree-decompositions of the graph. A graph which has a tree-decomposition of treewidth $O(1)$ is called a bounded treewidth graph.

FACT 1 (see [6]). For a bounded treewidth graph, a tree-decomposition of minimum treewidth can be found in linear time.

To state our approximation results on max-bisection we need the following definition.

DEFINITION 2.3. A real number α is said to be an approximation ratio for a maximization problem, or equivalently the problem is said to be approximable within a ratio α , if there is a polynomial time algorithm for the problem which always produces a solution of size at least α times the optimum. If a problem is approximable for arbitrary $\alpha < 1$, then it is said to admit a polynomial time approximation scheme (a PTAS for short).

An approximation ratio and a PTAS for a minimization problem are defined analogously.

3. Planar max-bisection is NP-hard. It is not difficult to observe that the known result of Garey, Johnson, and Stockmeyer asserting the NP-hardness of the maximum independent in planar cubic graphs [13] can be restated as follows:

Given a planar cubic graph G and an integer k , is there a $(k, n - k)$ -partition of G cutting at least $3k$ edges?

The only difference from max-bisection is that we have a $(k, n - k)$ -partition instead of a bisection. Jerrum has recently combined this observation with the idea of shifting the cut point in order to overcome the difference. The method of his proof follows a similar construction from the proof of NP-hardness of the problem of planar spin glass within a magnetic field (Problem P5) [4]. As a result, he obtained the following fact [18].

LEMMA 3.1. Let G be a planar cubic graph on n vertices, and let G' be the disjoint union of G with the complete bipartite graph $K_{2, n-2k+2}$, where $k \leq n/2$. G has an independent set of size k if and only if G' has a bisection of size at least $2n - k + 4$.

Proof. Suppose G has an independent set of size k . Form a $(k, n-k)$ -partition of G , where the set of cardinality k is an independent set in G , and the natural $(2, n-2k+2)$ -partition of $K_{2, n-2k+2}$. These two partitions yield a $(k+n-2k+2, n-k+2)$ -partition, i.e., a bisection of G' , cutting $3k+2(n-2k+2)$, i.e., $2n-k+4$ edges.

Suppose that G' has a bisection cutting at least $2n-k+4$ edges. The bisection yields an $(j, n-2k+2+2-j)$ -partition of the $K_{2, n-2k+2}$ subgraph and consequently an $(n-k-j+2, k+j-2)$ -partition of G . We may assume without loss of generality (w.l.o.g.) $j \leq n-2k+2+2-j$, which in turn implies $n-k-j+2 \geq k+j-2$. How many edges of $K_{2, n-2k+2}$ are cut by the bisection?

Clearly, the maximum $2(n-2k+2)$ can be achieved when $j=2$ and the $(j, n-2k+2+2-j)$ -partition of $K_{2, n-2k+2}$ is that bipartite one. If $j > 2$, then at least $j-2$ vertices out of the j vertices have to be on the larger side of $K_{2, n-2k+2}$, decreasing the maximum by at least $2(j-2)$. We conclude that if $j \geq 2$, at most $2(n-2k+2+2-j)$ edges can be cut. If $j=0, 1$, the number of edges that can be cut is clearly at most $j(n-2k+2)$.

Let U be the smaller set in the $(n-k-j+2, k+j-2)$ -partition of G , i.e., the one with $k+j-2$ vertices.

Consider the general case $j \geq 2$. By our assumptions, at least $2n-k+4-2(n-2k+4-j) = 3k+2j-4$ edges leave U . Suppose U has i internal edges (i.e., having both endpoints in U). Then by counting degrees, $3|U|-2i \geq 3k+2j-4$, i.e., $3(k+j-2)-2i \geq 3k+2j-4$, which yields $2i \leq j-2$. Thus, although U may not be an independent set itself, it has very few internal edges and must therefore contain a large independent set. Precisely, it must contain an independent set of size $|U|-2i \geq |U|-(j-2) = k$.

The cases $j=0, 1$ are easily seen to be infeasible: $|U|$ has size less than k but defines a partition of size at least $2n-k+4-(n-2k+2) > 2n-k+4-2(n-2k+2) = 3k$. \square

Lemma 3.1, combined with the NP-hardness of the maximum independent set for planar cubic graphs, immediately yields the NP-hardness of planar max-bisection.

THEOREM 3.2. *Planar max-bisection is NP-hard.*

4. Optimal partitions for graphs of bounded treewidth. Let G be a graph admitting a tree-decomposition $T = (I, F)$ of treewidth at most k for some constant k . By [9], one can easily modify T , without increasing its treewidth, such that one can see T as a rooted tree, with root $r \in I$, fulfilling the following conditions:

1. T is a binary tree.
2. If a node $i \in I$ has two children j_1 and j_2 , then $X_i = X_{j_1} = X_{j_2}$.
3. If a node $i \in I$ has one child j , then either $X_j \subset X_i$ and $|X_i \setminus X_j| = 1$, or $X_i \subset X_j$ and $|X_j \setminus X_i| = 1$.

We will assume in the remainder that such a modified tree-decomposition T of G is given.

For each node $i \in I$, let Y_i denote the set of all vertices in a set X_j with $j = i$ or j being a descendant of i in the rooted tree T . Our algorithm computes for each $i \in I$ an array $maxp_i$ with $O(2^k|Y_i|)$ entries. For each $l \in \{0, 1, \dots, |Y_i|\}$ and each subset S of X_i , the entry $maxp_i(l, S)$ is set to $\max_{S' \subseteq Y_i, |S'|=l, S' \cap X_i=S} |\{(v, w) \in E | v \in S' \text{ \& } w \in Y_i \setminus S'\}|$. In other words, $maxp_i(l, S)$ is set to the maximum number of cut edges in an $(l, |Y_i| - l)$ -partition of Y_i , where S and $X_i \setminus S$ are in the different sets of the partition and the set including S is of cardinality l . For convention, if such a partition is impossible, $maxp_i(l, S)$ will be set to $-\infty$.

The entries of the array are computed following the levels of the tree-decomposition T in a bottom-up manner. The following lemma shows how the array can be determined efficiently.

LEMMA 4.1.

- Let i be a leaf in T . Then for all $l \in \{0, 1, \dots, |X_i|\}$ and $S \subseteq X_i$, where $|S| = l$, $\max p_i(l, S) = |\{(v, w) \in E \mid v \in S, w \in X_i \setminus S\}|$. The remaining entries of $\max p_i$ are set to $-\infty$.
- Let i be a node with one child j in T . If $X_i \subseteq X_j$, then for all $l \in \{0, 1, \dots, |Y_i|\}$ and $S \subseteq X_i$, $\max p_i(l, S) = \max_{S' \subseteq X_j, S' \cap X_i = S} \max p_j(l, S')$.
- Let i be a node with one child j in T . If $X_j \cup \{v\} = X_i$, where $v \notin X_j$, then for all $l \in \{0, 1, \dots, |Y_i|\}$ and $S \subseteq X_i$, if $v \in S$, then $\max p_i(l, S) = \max p_j(l-1, S \setminus \{v\}) + |\{(v, s) \mid s \in X_i \setminus S\}|$; else $\max p_i(l, S) = \max p_j(l, S) + |\{(v, s) \mid s \in S\}|$.
- Let i be a node with two children j_1, j_2 in T , with $X_i = X_{j_1} = X_{j_2}$. For all $l \in \{0, 1, \dots, |Y_i|\}$ and $S \subseteq X_i$, $\max p_i(l, S) = \max_{l_1+l_2-|S|=l \& l_1 \geq |S| \& l_2 \geq |S|} (\max p_{j_1}(l_1, S) + \max p_{j_2}(l_2, S) - |\{(v, w) \in E \mid v \in S, w \in X_i \setminus S\}|)$.

It follows that computing an array $\max p_i$ on the basis of the arrays computed for the preceding level of T can be done in time $O(2^k |Y_i|^2)$. Consequently, one can compute the array $\max p_r$ for the root r of T in cubic time.

THEOREM 4.2. All maximum $(l, n-l)$ -partitions of a graph on n nodes given with a tree-decomposition of treewidth k can be computed in time $O(2^k n^3)$.

By substituting *min* for *max*, we can analogously compute all minimum $(l, n-l)$ -partitions of a graph with constant treewidth.

THEOREM 4.3. All minimum $(l, n-l)$ -partitions of a graph on n nodes given with a tree-decomposition of treewidth k can be computed in time $O(2^k n^3)$.

By Fact 1 we obtain the following corollary.

COROLLARY 4.4. All maximum and minimum $(l, n-l)$ -partitions of a bounded treewidth graph on n vertices can be computed in time $O(n^3)$.

Since a tree-decomposition of a planar graph on n vertices with treewidth $O(\sqrt{n})$ can be found in polynomial time by the planar separator theorem [7], we obtain also the following corollary.

COROLLARY 4.5. All maximum and minimum $(l, n-l)$ -partitions of a planar graph on n vertices can be computed in time $2^{O(\sqrt{n})}$.

5. A PTAS for max-bisection of an arbitrary planar graph. The authors of [19] observed that the requirements of the equal size of the vertex subsets in a two partition yielding a max-bisection makes the max-bisection problem hardly expressible as a maximum planar satisfiability formula. For this reason we cannot directly apply Khanna and Motwani's syntactic framework [20], yielding PTASs for several basic graph problems on planar graphs (e.g., max-cut). Instead, we combine the original Baker's method [3] with our algorithm for optimal maximum partitions on graphs of bounded treewidth via tree-type dynamic programming in order to derive the first PTAS for max-bisection of an arbitrary planar graph.

ALGORITHM 1.

input: a planar graph $G = (V, E)$ on n vertices and a positive integer k ;

output: $(1 - \frac{1}{k+1})$ -approximations of all maximum $(l, n-l)$ -partitions of G

1. Construct a plane embedding of G .
2. Set the level of a vertex in the embedding as follows: the vertices on the outer boundary have level 1, and the vertices on the outer boundary of the subgraph

obtained by deleting the vertices of level $i - 1$ have level i ; for convention extend the levels by k empty ones numbered $-k + 1, -k + 2, \dots, 0$.

3. For each level j in the embedding, construct the subgraph H_j of G induced by the vertices on levels $j, j + 1, \dots, j + k$.
4. For each level j in the embedding, set n_j to the number of vertices in H_j and compute all maximum $(l, n_j - l)$ -partitions of H_j .
5. For each i , $0 \leq i \leq k$, set G_i to the union of the subgraphs H_j , where $j \pmod{k + 1} = i$.
6. For each i , $0 \leq i \leq k$, compute all maximum $(l, n - l)$ -partitions of G_i by dynamic programming in a tree fashion; i.e., first compute all maximum partitions for pairs of “consecutive” H_j , where $j \pmod{k + 1} = i$, then for quadruples of such H_j , etc.
7. For each l , $1 \leq l < n$, output the largest among the maximum $(l, n - l)$ -partitions of G_i , $0 \leq i \leq k$.

LEMMA 5.1. *For each l , $1 \leq l < n$, Algorithm 1 outputs an $(l, n - l)$ -partition of G within $k/(k + 1)$ of the maximum.*

Proof. Let P be a maximum $(l, n - l)$ -partition of G .

We claim that for each edge e in P , there is at most one i , $0 \leq i \leq k$, such that e is not an edge of G_i . Let j be the maximum level of an endpoint of e . It follows that the other endpoint of e has level $j - 1$ and the endpoints of e belong to the subgraphs H_{j-k-1} and H_j , respectively. These are the only H subgraphs containing a single endpoint of e , and they both are part of the G_i subgraph where $j \pmod{k + 1} = i$.

By our claim, there is i' , $0 \leq i' \leq k$, such that $G_{i'}$ does not include at most $|P|/(k + 1)$ edges of P . It follows that a maximum $(l, n - l)$ -partition of such a $G_{i'}$ cuts at least $k|P|/(k + 1)$ edges. Algorithm 1 outputs an $(l, n - l)$ -partition of G cutting at least as many edges as a maximum $(l, n - l)$ -partition of $G_{i'}$. \square

LEMMA 5.2. *Algorithm 1 runs in $O(k2^{3k+2}n^3)$ time.*

Proof. The time complexity of the algorithm is dominated by that of steps 4 and 6.

The subgraphs H_j of G are so-called $(k + 1)$ -outerplanar graphs and have bounded treewidth $3k + 2$ [7]. Hence, for a given i , $0 \leq i \leq k$, all maximum $(l, n_j - l)$ -partitions of H_j , where $j \pmod{k + 1} = i$, can be computed in time $O(2^{3k+2}n^3)$ by Theorem 4.2, the pairwise disjointness of the subgraphs, and $j \leq n$. It follows that the whole step 4 can be implemented in time $O(k2^{3k+2}n^3)$.

In step 6, a maximum $(l, n' - l)$ -partition of the union of 2^{q+1} “consecutive” H_j ’s satisfying $j \pmod{k + 1} = i$ on n' vertices can be determined on the basis of appropriate maximum partitions of its two halves, each being the union of 2^q of the H_j ’s, in time $O(n)$. Hence, since $l \leq n'$ and the number of nodes in the dynamic programming tree is $O(n)$, the whole step 6 takes $O(kn^3)$ time. \square

THEOREM 5.3. *Algorithm 1 yields a PTAS for all maximum $(l, n - l)$ -partitions of a planar graph.*

COROLLARY 5.4. *The problem of max-bisection on planar graphs admits a PTAS.*

6. Observations on min-bisection for planar graphs. We can easily obtain an analogous PTAS for min-bisection of planar graphs in the very special case when the size of min-bisection is $\Omega(n)$. Simply, at least one of the subgraphs G_i of G misses at most $|E|/(k + 1)$ edges of G . Therefore, the number of edges cut by a min-bisection of such a G_i can increase at most by $|E|/(k + 1)$ in G . By picking k sufficiently large we can guarantee an arbitrarily close approximation of min-bisection in G .

In fact, we can obtain even a slightly stronger result on min-bisection for planar graphs by observing that our method runs in polynomial time even for nonconstant k (up to $O(\log n)$), provided that a tree-decomposition of graphs with treewidth equal to such a k can be determined in polynomial time. At present, the best tree-decomposition algorithms have the leading term k^k [8], so we can set k to $O(\log n / \log \log n)$, keeping the polynomial time performance of our method. In this way, we obtain the following theorem.

THEOREM 6.1. *The min-bisection problem on planar graphs in which the size of min-bisection is $\Omega(n \log \log n / \log n)$ admits a PTAS.*

Observe that the presence of large degree vertices in a planar graph can cause the large size of min-bisection, e.g., in a star graph. For bounded-degree planar graphs, the size of min-bisection is $O(\sqrt{n})$ by the following argument.

The planar separator theorem yields a linear-time method of disconnecting a planar graph into two subgraphs, each having at least one third of its vertices, by removing $O(\sqrt{n})$ vertices [21]. For a planar graph of maximum degree d , construct a separator tree by applying the planar separator theorem recursively. Next, find a path in the tree from the root down to the median leaf. By deleting the edges incident to the vertex separators along the path and additionally $O(1)$ edges, we can easily halve the set of vertices of the graph such that none of the remaining edges connects a pair of vertices from the opposite halves. The number of deleted edges at the i th level of the tree is $O(d\sqrt{n}(\frac{2}{3})^{i/2})$. Thus, the total number of deleted edges is $O(d\sqrt{n})$. In fact, we do not have to construct the whole separator tree, but just the path, and this can be easily done in time $O(n \log n)$ [21].

THEOREM 6.2. *For a planar graph on n vertices and maximum degree d , a bisection of size $O(d\sqrt{n})$ can be found in time $O(n \log n)$.*

Clearly, if a graph has an $O(1)$ -size bisection, it can be found by exhaustive search in polynomial time. We conclude that at present we have efficient methods for at least $O(1)$ -approximation of min-bisection in planar graphs if its size is either $\Omega(n \log \log n / \log n)$ or $O(1)$, or it is $O(\sqrt{n})$ and the maximum degree is constantly bounded. These observations suggest that a substantial improvement of the logarithmic approximation factor for min-bisection on planar graphs given in [11] might be possible.

7. PTAS for max-bisection of a unit disk graph. In this section we design a PTAS for max-bisection of unit disk graphs, another important class of graphs defined by the geometric conditions on a plane.

Recall that an undirected graph G is a unit disk graph if its vertices can be put in one-to-one correspondence with disks of equal radius in the plane in such a way that two vertices are joined by an edge if and only if the corresponding disks intersect. Tangent disks are considered to intersect. We may assume w.l.o.g. that the radius of each disk is one. Since the recognition problem for the unit disk graph is NP-hard, we shall also assume that a geometric representation of the graph is given as input.

Our technique works in a similar way as in the case for planar graphs. The input graph G is divided into families of subgraphs $H_{i,j}$ using the ideas of Hunt et al. given in [17]. Next, either an optimal or approximative solution to all $(l, n_{i,j} - l)$ -partitions of every subgraph $H_{i,j}$, where $n_{i,j}$ denotes the number of vertices in $H_{i,j}$, can be computed. If the number of vertices $n_{i,j}$ is smaller than a constant c , then we can compute an optimal solution in constant time. If the number $n_{i,j}$ is larger than a constant c , then we use the methods given in [2]. We show below that in such a case the underlying graph is dense; i.e., it has at least $an_{i,j}^2$ edges for a constant $a > 0$. Via

tree-type dynamic programming these solutions are used to obtain an overall solution for G .

In order to divide the graph G , we impose a grid of horizontal and vertical lines on the plane that are 2 apart of each other. The v th vertical line, $-\infty < v < \infty$, is at $x = 2v$. The h th horizontal line, $-\infty < h < \infty$, is at $y = 2h$. We say that the v th vertical line has index v and that the h horizontal line has index h . Further, we denote the vertical strip between the v th and the $(v + 1)$ th vertical line as the strip with index v and as the analogue for the horizontal strip between the h th and the $(h + 1)$ th horizontal line.

Each vertical strip is left-closed and right-open, and each horizontal strip is closed at the top and open at the bottom. A disk is said to lie in a given strip if its center lies in that strip. Note that every disk lies in exactly one horizontal and vertical strip.

For a fixed k , consider the subgraph $H_{i,j}$ of G , $-\infty < i, j < \infty$, induced by the disks that lie in the intersection of the horizontal strips $i, i + 1, \dots, i + k$ and the vertical strips $j, j + 1, \dots, j + k$. Let $n_{i,j}$ be the number of vertices of $H_{i,j}$. By a packing argument it can be shown that for fixed $k > 0$, the size of a maximum independent set of such a subgraph is at most $4(k + 3)^2/\pi$. The area of the square (given by the intersection of the $(k + 1)$ horizontal and $(k + 1)$ vertical strips) is $4(k + 1)^2$. Since a disk can lie close to or at the border of a horizontal strip i and $i + k$ and at the border of a vertical strip j and $j + k$, the total area that an independent set of disks (of subgraph $H_{i,j}$) can cover is at most $4(k + 3)^2$. Since the area of each disk is π , the size of an independent set is at most $4(k + 3)^2/\pi$.

LEMMA 7.1. *If $n_{i,j} > 8(k + 3)^2/\pi$, then the subgraph $H_{i,j}$ of G is dense.*

Proof. Partition the vertex set of $H_{i,j}$ successively into maximal independent sets by determining a maximal independent set I_1 , remove its vertices, and again determine a maximal independent set I_2 , and so on. As described above, the number of independent sets is at least $\pi n_{i,j}/4(k + 3)^2$. Since each I_j is maximal, there is at least one edge from a vertex of I_j to every $I_{j'}$, $j < j'$. If we understand the set of independent sets as a complete graph on $\pi n_{i,j}/4(k + 3)^2$ vertices, it follows that $H_{i,j}$ has at least $n_{i,j}^2 \pi^2/64(k + 3)^4$ edges for $n_{i,j} \geq 8(k + 3)^2/\pi$. This implies that $H_{i,j}$ is dense. \square

COROLLARY 7.2. *If $n_{i,j} > 8(k + 3)^2/\pi$, then the size of a maximum bisection of $H_{i,j}$ is $\Omega(n_{i,j}^2)$.*

Proof. Partition the vertex set of $H_{i,j}$ as before and use the maximum independent sets to build up the sets of the bisection. Since all independent sets are maximal, there are at least $n_{i,j}^2 \pi^2/16^2(k + 3)^4$ edges between the sets of bisection. \square

The main idea of our algorithm is that each subgraph $H_{i,j}$ (corresponding to the $(k + 1) \times (k + 1)$ square) either has at most a constant number of vertices (case $n_{i,j} \leq 8(k + 3)^2/\pi$) or is dense and has a large bisection (case $n_{i,j} > 8(k + 3)^2/\pi$; see Lemma 7.1 and Corollary 7.2 above). In the former case, the problem can be solved optimally, while in the later case, we use a PTAS of [2] to obtain an approximately optimal solution. Let $c = 8(k + 3)^2/\pi$.

ALGORITHM 2.

input: a unit disk graph $G = (V, E)$ specified by a set V of disks in the plane and the coordinates of their centers and a positive integer k ;

output: $(1 - \frac{1}{k+1})^2(1 - \delta)$ -approximations of maximum bisection of G

1. Divide the plane by imposing a grid of width two.
2. Construct the subgraphs $H_{i,j}$ of G as described above.
3. For each i and each j , set $n'_{i,j}$ to the number of vertices in $H_{i,j}$ and compute all

$(l, n'_{i,j} - l)$ -partitions of $H_{i,j}$ either approximatively (for $n'_{i,j} > c$) or optimally (for $n'_{i,j} \leq c$).

4. For each r and s , $0 \leq r, s \leq k$, set $G_{r,s}$ to the union of the subgraphs $H_{i,j}$, where $i \pmod{k+1} = r$ and $j \pmod{k+1} = s$.
5. For each r and s , $0 \leq r, s \leq k$, set $n_{r,s}$ to the number of vertices in $G_{r,s}$ and compute a bisection of $G_{r,s}$ within $(1 - \delta)$ of its maximum by dynamic programming in a tree fashion. Therefore, enumerate the subgraphs in increasing order of the sum $i + j$ and compute all partitions of pairs of “consecutive” $H_{i,j}$, respectively, to this ordering on the basis of the computed partitions, then for quadruples of such $H_{i,j}$, etc.
6. Output the largest bisection of $G_{r,s}$, $0 \leq r, s \leq k$.

If $n_{i,j} \leq c = 8(k+3)^2/\pi$, we can find all the maximum $(l, n_{i,j} - l)$ -partitions of the subgraph $H_{i,j}$ in constant time by enumerating all possibilities. Otherwise (for $n_{i,j} > c = 8(k+3)^2/\pi$) the problem is solvable approximatively in polynomial time by solving the following polynomial integer program:

$$(7.1) \quad \text{maximize} \quad \sum_{\{i,j\} \in E(H_{i,j})} x_i(1-x_j) + x_j(1-x_i)$$

$$(7.2) \quad \text{subject to} \quad \sum x_i = l,$$

$$(7.3) \quad x_i \in \{0, 1\}, \quad i = (1, \dots, n_{i,j}).$$

This program can be solved by the use of Theorem 1.10 in [2] within an error of at most $\epsilon n_{i,j}^2$, which also satisfies the linear constraint (7.2) of the program within an additive error of $O(\epsilon \sqrt{n_{i,j} \log n_{i,j}})$. In order to get a subset of size l , we move at most $\epsilon \sqrt{n_{i,j} \log n_{i,j}}$ in or out. This affects the number of edges included in the partition by at most $\epsilon n_{i,j} \sqrt{n_{i,j} \log n_{i,j}} \leq \epsilon n_{i,j}^2$. Hence we can compute a maximum $(l, n_{i,j} - l)$ -partition of a subgraph $H_{i,j}$ that has more than c vertices within an additive error of $2\epsilon n_{i,j}^2$ of the maximum.

LEMMA 7.3. *Algorithm 2 outputs a bisection of G within $(1 - \frac{1}{k+1})^2(1 - \delta)$ of the maximum.*

Proof. Let P be a maximum bisection of G . For each edge $e \in P$ and a fixed r , $0 \leq r \leq k$, there is at most one s , $0 \leq s \leq k$, such that e crosses a vertical line whose index modulo $k+1$ is s . Analogously, there is for each $e \in P$ and a fixed s , $0 \leq s \leq k$, at most one r , $0 \leq r \leq k$, such that e crosses a horizontal line whose index modulo $k+1$ is r . Consequently, there is a pair (r, s) , $0 \leq r, s \leq k$, such that a maximum $(l, n - l)$ -partition of $G_{r,s}$ cuts at least $(1 - \frac{1}{k+1})^2 |P|$ edges.

By Corollary 7.2, the size of maximum bisection of the subgraph $G'_{r,s}$ of $G_{r,s}$ that consists of all $H_{i,j}$ with more than $c = 8(k+3)^2/\pi$ vertices is at least $\sum_{n_{i,j} > c} dn_{i,j}^2$, where $d = \pi^2/16^2(k+3)^4$. Choosing $\epsilon = \delta d/2$, the error $2\epsilon n_{i,j}^2 = \delta dn_{i,j}^2$ caused by the solutions of the polynomial integer programs for the subgraphs $H_{i,j}$ of $G'_{r,s}$ is at most a δ fraction of an optimum solution of maximum bisection for $G'_{r,s}$. Since the partitions for each $H_{i,j}$ with at most c vertices are computed optimally, we obtain a bisection of $G_{r,s}$ within $(1 - \delta)$ of the maximum.

Thus Algorithm 2 outputs a bisection of G within $(1 - \frac{1}{k+1})^2(1 - \delta)$ of the maximum. \square

THEOREM 7.4. *The problem of max-bisection on unit disk graphs admits a PTAS.*

The same approach can be used to obtain a PTAS for the maximum bisection problem in geometric intersection graphs, both of other regular polygons and also of regular geometric objects in higher dimensions.

Acknowledgments. We are grateful to Mark Jerrum for providing the proof of NP-hardness of planar max-bisection in exact setting and agreeing on presenting it in this paper. We thank also Andreas Björklund, Hans Bodlaender, Uri Feige, Miroslaw Kowaluk, Mike Langberg, and Monique Laurent for many stimulating remarks and discussions.

REFERENCES

- [1] S. ARNBORG, *Efficient algorithms for combinatorial problems on graphs with bounded decomposability—A survey*, BIT, 25 (1985), pp. 2–23.
- [2] S. ARORA, D. KARGER, AND M. KARPINSKI, *Polynomial time approximation schemes for dense instances of NP-hard problems*, J. Comput. System Sci., 58 (1999), pp. 193–210.
- [3] B. S. BAKER, *Approximation algorithms for NP-complete problems on planar graphs*, J. Assoc. Comput. Mach., 41 (1994), pp. 153–180.
- [4] F. BARAHONA, *On the computational complexity of Ising spin glass models*, J. Phys. A., 15 (1982), pp. 3241–3253.
- [5] H. L. BODLAENDER, *A tourist guide through treewidth*, Acta Cybernet., 11 (1993), pp. 1–23.
- [6] H. L. BODLAENDER, *A linear-time algorithm for finding tree-decompositions of small treewidth*, SIAM J. Comput., 25 (1996), pp. 1305–1317.
- [7] H. L. BODLAENDER, *A partial k -arboretum of graphs with bounded treewidth*, Theoret. Comput. Sci., 209 (1998), pp. 1–47.
- [8] H. L. BODLAENDER, *private communication*, 2000.
- [9] H. L. BODLAENDER AND K. JANSEN, *On the complexity of the Maximum Cut problem*, Nordic J. Comput., 7 (2000), pp. 14–31.
- [10] U. FEIGE, M. KARPINSKI, AND M. LANGBERG, *A note on approximating Max-Bisection on regular graphs*, Inform. Process. Lett., 79 (2001), pp. 181–188.
- [11] U. FEIGE AND R. KRAUTHGAMER, *A polylogarithmic approximation of the minimum bisection*, SIAM J. Comput., 31 (2002), pp. 1090–1118.
- [12] A. FRIEZE AND M. JERRUM, *Improved approximation algorithms for MAX k -CUT and MAX BISECTION*, Algorithmica, 18 (1997), pp. 67–81.
- [13] M. R. GAREY AND D. S. JOHNSON, *Computers and Intractability. A Guide to the Theory of NP-completeness*, Freeman, San Francisco, 1979.
- [14] M. X. GOEMANS AND D. P. WILLIAMSON, *Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming*, J. Assoc. Comput. Mach., 42 (1995), pp. 1115–1145.
- [15] F. HADLOCK, *Finding a maximum cut of a planar graph in polynomial time*, SIAM J. Comput., 4 (1975), pp. 221–225.
- [16] E. HALPERIN AND U. ZWICK, *A unified framework for obtaining improved approximation algorithms for maximum graph bisection problems*, Random Structures Algorithms, 20 (2002), pp. 382–402.
- [17] H. B. HUNT, M. V. MARATHE, V. RADHAKRISHNAN, S. S. RAVI, D. S. ROSENKRANTZ, AND R. E. STEARNS, *NC-approximation schemes for NP- and PSPACE-hard problems for geometric graphs*, J. Algorithms, 26 (1998), pp. 238–274.
- [18] M. JERRUM, *private communication*, 2000.
- [19] M. KARPINSKI, M. KOWALUK, AND A. LINGAS, *Approximation algorithms for max-bisection on low degree regular graphs*, Fund. Inform., 62 (2004), pp. 369–375.
- [20] S. KHANNA AND R. MOTWANI, *Towards a syntactic characterization of PTAS*, in Proceedings of the 28th Annual ACM Symposium on the Theory of Computing, 1996, pp. 329–337.
- [21] R. J. LIPTON AND R. E. TARJAN, *A separator theorem for planar graphs*, SIAM J. Appl. Math., 36 (1979), pp. 177–189.
- [22] N. ROBERTSON AND P. D. SEYMOUR, *Graph minors. II. Algorithmic aspects of tree-width*, J. Algorithms, 7 (1986), pp. 309–322.
- [23] Y. YE, *A 0.699-approximation algorithm for Max-Bisection*, Math. Program., 90 (2001), pp. 101–111.

MINIMUM-WEIGHT SPANNING TREE CONSTRUCTION IN $O(\log \log n)$ COMMUNICATION ROUNDS*

ZVI LOTKER[†], BOAZ PATT-SHAMIR[†], ELAN PAVLOV[‡], AND DAVID PELEG[§]

Abstract. We consider a simple model for overlay networks, where all n processes are connected to all other processes, and each message contains at most $O(\log n)$ bits. For this model, we present a distributed algorithm which constructs a minimum-weight spanning tree in $O(\log \log n)$ communication rounds, where in each round any process can send a message to every other process. If message size is $\Theta(n^\epsilon)$ for some $\epsilon > 0$, then the number of communication rounds is $O(\log \frac{1}{\epsilon})$.

Key words. minimum-weight spanning tree, distributed algorithms

AMS subject classifications. 05C05, 05C85, 68Q22, 68Q25, 68R10

DOI. 10.1137/S0097539704441848

1. Introduction. A minimum-weight spanning tree (MST) is one of the most useful distributed constructs, as it minimizes the cost associated with global operations such as broadcasts and convergecasts. This paper presents an MST construction algorithm that works in $O(\log \log n)$ communication rounds, where in each round each process can send $O(\log n)$ bits to every other process (intuitively allowing each message to contain the identity and weight of only a constant number of edges). Our result shows that an MST can be constructed with little communication: throughout the execution of the algorithm, each pair of processes exchanges at most $O(\log n \log \log n)$ bits; the overall number of bits sent is $\Theta(n^2 \log n)$, which is optimal. The algorithm extends to larger message sizes, in the sense that the number of communication rounds is $O(\log \frac{1}{\epsilon})$ if each message can contain n^ϵ bits for some $\epsilon > 0$. Note that if messages are not restricted in size, then the MST can be trivially constructed in a single round of communication: each process sends all its information to all its neighbors, allowing each node to locally compute the MST.

The number of communication rounds dominates the time complexity in situations where latency is high and bandwidth is scarce. This may be the situation in some *overlay networks*. Briefly, the idea in overlay networks is to think of the underlying communication network (e.g., the Internet) as a “black box” that provides reliable point-to-point communication. On top of that network run distributed applications. This approach (whose precursor is the Internet’s “end-to-end argument” [13]) is different from classical distributed models, where processes reside in networks nodes (i.e., switches or routers), and thus their implementation would require using low-level communication. Rather, the pragmatic view now is that distributed applica-

*Received by the editors February 11, 2004; accepted for publication (in revised form) April 27, 2005; published electronically September 8, 2005. A preliminary version of this work appeared in *Proceedings of the 15th Annual ACM Symposium on Parallelism in Algorithms and Architectures*, San Diego, CA, 2003.

<http://www.siam.org/journals/sicomp/35-1/44184.html>

[†]Department of Electrical Engineering, Tel Aviv University, Tel Aviv 69978, Israel (zvilo@eng.tau.ac.il, boaz@eng.tau.ac.il).

[‡]Department of Computer Science, The Hebrew University, Jerusalem 91904, Israel (elan@cs.huji.ac.il).

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

tions create their own overlay network by choosing which pairs of local processes will communicate directly according to various criteria. The concept of overlay networks is central to areas such as multicast or content distribution networks (see, e.g., [8] and the references therein), peer-to-peer systems (for example, Chord [14]), and others.

1.1. Related work. Spanning tree construction is well studied as a sequential optimization problem (see, e.g., [15, 9]). Distributed MST constructions are presented in [6, 3] (and see the references in [11]). These classical distributed algorithms are oriented towards minimizing the total number of messages in general networks, and their time complexity is inherently $\Omega(\log n)$, even when run on fully connected graphs. The model we use in this paper is a special case of the model studied in [7, 12, 10]: in these papers, each message has $O(\log n)$ bits, but the fully connected graph is not directly considered. The best previously known upper bound for fully connected graphs in this model is $O(\log n)$ communication rounds. This bound holds also for graphs of diameter 2 [10]. (It is known that the number of rounds jumps at least to $\Omega(n^{1/4})$ when the diameter of the network is 3 or more [10, 12].)

The parallel time complexity of MST construction depends on the particular architecture considered, but we are not aware of any sublogarithmic time algorithm that uses small messages. For the PRAM model, there are quite a few $O(\log n)$ algorithms, including a deterministic one for the CRCW model [4] and a randomized one for the EREW model [5]. Adler et al. [1] study the total number of bits that must be communicated in the course of an MST construction problem under various parallel architectures. For our model, their results imply that the worst-case number of bits that need to be communicated throughout the execution of the algorithm is $\Omega(n^2 \log n)$.

1.2. System model. In the underlying formal model, the system is represented by a complete n -node weighted undirected graph $G = (V, E, \omega)$, where $\omega(e)$ denotes the weight of edge $e \in E$. Each node has a distinct ID of $O(\log n)$ bits. Each node knows all the edges incident to it (and hence, since the graph is a clique, each node knows about all other nodes in the system). An execution of the system proceeds in asynchronous steps: in a “receive” step, a node receives some of the messages sent to it in previous steps. In a “send” step, a node makes a local computation and sends messages to the other nodes in the system. Each message may be different, and we require that each message contains at most $O(\log n)$ bits. (The results are extended to larger message sizes in section 4.) We assume that messages may be delayed arbitrarily but are never lost or corrupted. The time complexity of an algorithm in the asynchronous model is measured by normalizing the scale so that the longest message delivery time is one unit.

Simplification: The synchronous model. In the synchronous model, computation advances in global rounds, where in each round processes send messages, receive them, and do some local computation. This model is much more convenient as a programming mode. Fortunately, since we assume that the system is reliable, we may apply a *synchronizer* that allows us to present the algorithm in the synchronous model. Specifically, we use the α synchronizer of Awerbuch [2]. Let us outline the idea briefly. Assume that we have an algorithm SA for the synchronous model. The execution in the asynchronous model is done as follows. A process starts the next round only after receiving a special “proceed” message from a distinguished node v^* (say, the node with the lowest ID in the system). It then sends messages according to SA . For each SA message received, the receiver node sends an “ack” message back to the sender; when a sender has received acknowledgements to all the messages it

sent, the sender forwards a “safe” message to v^* ; when v^* receives “safe” messages from all nodes in the system, it sends a “proceed” message to all other nodes, which may then send their SA messages of the next round. Note that since we assume that the graph is fully connected, this transformation incurs only a constant blowup in the message complexity and in time complexity. We shall henceforth use the synchronous model, but we emphasize that the algorithm works in the asynchronous model using the simple synchronizer described above.

1.3. The MST construction problem. We assume that in the initial state, the input to each node $v \in V$ consists of the weights of all its incident edges $\omega(v, u)$ for all $u \in V \setminus \{v\}$. Edge weights are assumed to be integers that can be represented using $O(\log n)$ bits. Without loss of generality, we assume that all the edge weights are distinct (otherwise we can break ties by node IDs), and hence the MST is unique. When our algorithm halts, all nodes know the full list of all $n - 1$ edges in the MST of G .

2. Algorithm description. In this section we describe the algorithm. In section 2.1 we give an overview of the main ideas. In section 2.2 we specify the main algorithm, and in sections 2.3 and 2.4 we specify local subroutines used by the main algorithm.

2.1. Overview. The algorithm operates in phases: Each phase takes $O(1)$ rounds, and there are at most $O(\log \log n)$ phases. At the end of each phase $k \geq 0$, the nodes of G are partitioned into disjoint clusters $\mathcal{F}^k = \{F_1^k, \dots, F_{m_k}^k\}$, $\bigcup_i F_i^k = V$. For each cluster $F \in \mathcal{F}^k$, the algorithm selects also a spanning subtree $T(F)$. The partition \mathcal{F}^k and the corresponding subtree collection $\mathcal{T}^k = \{T(F) \mid F \in \mathcal{F}^k\}$, including the weights of the edges in those subtrees, are known to every vertex in the graph. (For notational consistency, we think of the initial situation at the beginning of phase 1 as the end of an “imaginary” phase 0, with each node forming a singleton cluster, i.e., $\mathcal{F}^0 = \{F_1^0, \dots, F_n^0\}$, where $F_i^0 = \{v_i\}$ for every $1 \leq i \leq n$.)

Define a *fragment* to be a connected subtree of the MST. For a set of nodes $F \subseteq V$, denote by $\mathcal{T}(F)$ the subgraph of the MST induced by F . With these notations, we can state the following invariant, satisfied by the algorithm at the end of each phase $k \geq 0$:

$\mathcal{T}(F) = T(F)$ for every cluster F ; namely, the spanning subtree selected for F is a fragment.

In our model, it is easy for the nodes of each cluster to learn, in constant time, the lightest edge to every other cluster. Hence, intuitively, it is possible to “contract” each cluster C into a vertex v_C , thus creating a smaller logical graph \hat{G} , and continue working on this logical graph. (In practice, each real vertex belonging to some cluster C knows the weight of the edge connecting its vertex v_C to every other vertex in \hat{G} . The operations of each vertex v_C of the logical graph \hat{G} are carried out by the real vertices belonging to the cluster C , or by a single representative called the *leader* of C , denoted $\ell(C)$.) This enables us to simulate the usual “fragment growing” MST construction process for \hat{G} , based on examining the edges one by one in increasing order of weight and including in the MST each inspected edge that is the *minimum-weight outgoing edge (MWOE)* of its fragment. This can be done in $O(\log n)$ time.

To reduce the time complexity to $O(\log \log n)$, it is necessary to speed up the process by making the cluster sizes grow quadratically in each phase. The main idea used for achieving this growth rate is the following. Essentially, we would like to provide every vertex v_C in the logical graph \hat{G} with information about additional

edges in \hat{G} , beyond its own. In particular, if we were somehow able to let every vertex v_C learn the *entire* topology of \hat{G} , then we could finish the MST construction for \hat{G} in a single step by asking each vertex in the graph to compute the MST locally. Unfortunately, such information exchange seems to require too much time. On the positive side, denoting the minimum cluster size by N , it is possible for the (N or more) members of each cluster to inform a distinguished vertex v^* of the graph, in constant time, of the N lightest edges connecting their cluster to other clusters, by appropriately sharing the workload of this task among them. (For concreteness, we assume that v^* is the node with the smallest ID in the system.)

Subsequently, we now face a special subtask of the MST construction problem to solve in v^* . This node now has a partial picture of the logical graph \hat{G} , consisting of all the vertices v_C but only some of the edges connecting them, particularly the N lightest edges emanating from each vertex of \hat{G} (to N other vertices). It is now necessary to perform (locally) as many *legal* “fragment merging” steps as possible on the basis of this information. That is, we would like to sort the edges known to us by increasing order of weight, examine them one by one, and add edges that are the MWOE of one of the two fragments they connect, so long as we can be sure of that fact. So the question becomes: When is it “dangerous” to continue the merging steps in the absence of information about the weights of the edges unknown to us?

The answer to this question is that it is perfectly safe to continue merging a fragment F (in the logical graph \hat{G}), so long as for each vertex v_C in F we have still not inspected *at least one* of its N lightest edges (which is known to us by assumption). However, once we have already inspected all the edges of some vertex v_C in the fragment F , it becomes dangerous to continue attempting to merge the fragment over edges known to us, as it is possible that the true MWOE of F is the $(N + 1)$ st lightest edge emanating from v_C , which is not known to us (yet is lighter than any edge emanating from C that we do know of at this moment).

The crucial observation is that this “safety rule” still allows us to grow each of the fragments to contain at least $N + 1$ vertices of \hat{G} . This means that the clusters of the next phase will be of minimum size $\Omega(N^2)$.

An interesting observation is that even when we can no longer identify the MWOE of some fragment F , we may still be able to safely merge F with some other fragment F' . This may still be legitimate if we can ascertain that the edge connecting F and F' is the MWOE of F' .

Finally, after constructing locally the new fragments, v^* sends out the identity of the edges added to the chosen set. This can be done in constant time by letting v^* send each edge to a different intermediate node, which will broadcast that edge to all other nodes.

2.2. The main algorithm. In the algorithm, whenever a node is instructed to send a message containing the edge $e = (u, v)$, this should be interpreted as a message including the IDs of its two endpoints, $ID(u)$ and $ID(v)$, as well as the edge weight $\omega(e)$.

We now describe the steps taken in phase k for all $1 \leq k \leq \log \log n$. Let v^* denote the node whose ID is minimal among all nodes in the graph.

Throughout, the algorithm is illustrated on the 16-vertex complete graph K_{16} with weights as depicted in Figure 1. Note that in this case there are only two phases. The flow of the algorithm is illustrated in Figure 2. In the first column, the fragment leaders are marked by horizontal stripes. Note that in the first phase all the nodes are leaders, whereas in the second phase only half of the nodes are leaders. The

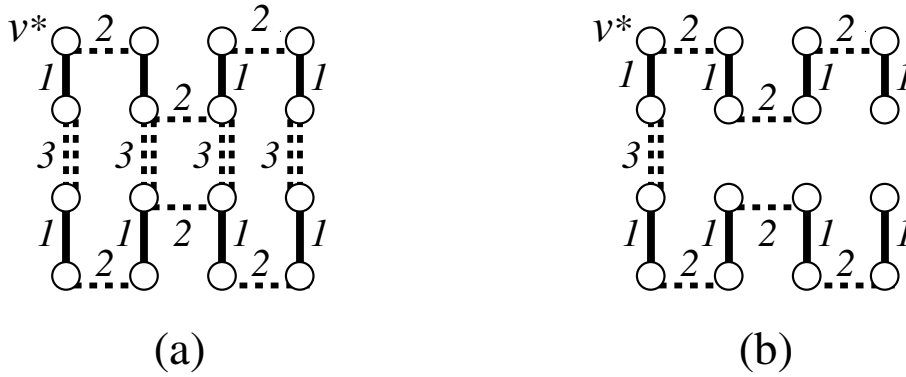


FIG. 1. The example graph K_{16} : (a) The edge weights (weight 1 is a solid line, 2 is a dashed line, and 3 is a double-dashed line). Edges not shown in the figure have weight 4; hence they do not participate in the MST. (b) The resulting MST.

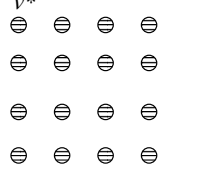
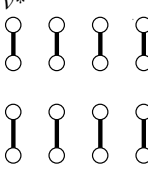
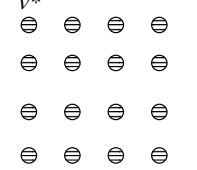
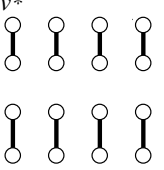
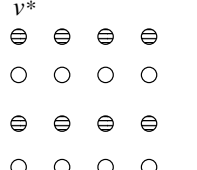
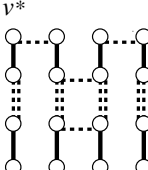
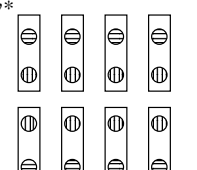
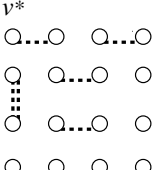
Phase	Fragment leaders	Selected edges	Edge guardians	Information sent by v^*
1	v^* 	v^* 	v^* 	v^* 
2	v^* 	v^* 	v^* 	v^* 

FIG. 2. An illustration of the execution of the algorithm on the example graph K_{16} .

second column shows the selected edges. In the first phase each fragment chooses one edge, while in the second phase each fragment chooses two edges, with the cheapest edge of each fragment denoted by a single-dashed line and the second cheapest edge denoted by a double-dashed line. The third column shows the guardian of each of the selected edges: the horizontally striped nodes are the guardians of cheapest edges and the vertically striped nodes are the guardians of the second cheapest edges. The last column shows the new edges that node v^* adds to the MST.

Phase k : Code for node v in cluster F of size $N = |F|$

Input: A set of chosen edges. The set of connected components defined by this set is the set of clusters \mathcal{F}^{k-1} . For each cluster $F' \in \mathcal{F}^{k-1}$, $\ell(F')$ is the node with the minimal ID in F' .

1. (a) Compute the minimum-weight edge $e(v, F')$ that connects v to (any node of) F' for all clusters $F' \neq F$.
 (b) Send $e(v, F')$ to $\ell(F')$ for all clusters $F' \neq F$.
 2. **If $v = \ell(F)$ then**
 - (a) Using the messages received from step 1, compute the lightest edge between F' and F for every other cluster F' .
 - (b) Perform (locally) Procedure **Cheap_Out** (described below), which does the following:
 - It selects a set $\mathcal{A}(F)$ containing the N cheapest MWOEs that go out of F to $N = |F|$ distinct clusters.
 - It appoints for each such edge e a *guardian* node $g(e)$ in F , ensuring that each node in F is appointed as guardian to at most one edge.
 3. Let $e' \in \mathcal{A}(F)$ be the edge for which v was appointed as guardian, i.e., such that $g(e') = v$. Send e' to v^* , the node with the minimal ID in the graph.
 (At the end of this step, v^* knows all the edges in the set $\mathcal{A} = \bigcup_{F' \in \mathcal{F}^{k-1}} \mathcal{A}(F')$.)
 4. **If $v = v^*$ then**
 - (a) Perform (locally) Procedure **Const_Frags**. This procedure (described below) computes E^k , the new set of edges to add.
 - (b) For each edge $e \in E^k$, send a message to $g(e)$.
 5. **If v receives a message from v^* that $e \in E^k$** , then v sends e to all nodes in the graph.
 6. Each node adds all edges in E^k and computes \mathcal{F}^k .
-

2.3. Procedure Cheap_Out. The local procedure **Cheap_Out** is invoked by cluster leaders in each phase, and it operates as follows at the leader of cluster F with $|F| = N$ at phase k .

Input: Cheapest edge $e(F, F')$ for every $F' \in \mathcal{F}^{k-1}$.

1. Sort the input edges in increasing order of weight.
 2. Let $\mu = \min\{N, |\mathcal{F}^{k-1}| - 1\}$.
 3. Define $\mathcal{A}(F)$ to be the first μ edges in the sorted list.
 4. Sort the nodes of F by increasing order of ID.
 5. Appoint the i th node of F as the guardian of the i th edge added to $\mathcal{A}(F)$.
 6. For each node $u \in F$: send the edge to which u is appointed.
-

2.4. Procedure Const_Frags. The local procedure **Const_Frags** is invoked only by the distinguished node v^* , and it operates as follows. It receives as input the initial partition \mathcal{F}^{k-1} , the spanning subtree collection \mathcal{T}^{k-1} , and the set of edges for inspection, \mathcal{A} . Its output is a set of edges E^k , which defines a new partition \mathcal{F}^k and its spanning subtree \mathcal{T}^k : the edge set of \mathcal{T}^k is the union of the set of edges in \mathcal{T}^{k-1} with the set E^k , and \mathcal{F}^k is the set of connected components of \mathcal{T}^k .

The procedure operates in two stages. In the first stage, it contracts the input clusters into vertices, thus creating a *logical graph* \hat{G} , partitions this logical graph into “superclusters,” and constructs a spanning subtree for each such supercluster. In the second stage, the procedure transforms the superclusters and spanning subtrees constructed for \hat{G} into clusters and spanning subtrees for the original graph G .

We now continue with a more detailed description of the two stages. The first stage operates as follows. The procedure starts by creating the logical graph $\hat{G} = (\hat{V}, \hat{E})$, where each input cluster is viewed as a vertex, namely, $\hat{V} = \mathcal{F}^{k-1}$. The edge set \hat{E} consists of the logical edges corresponding to the edges of the set \mathcal{A} . Set the logical edge corresponding to $e = (u, w)$ to be $X(e) = (F, F')$, where $u \in F$ and $w \in F'$. Then $\hat{E} = \{X(e) \mid e \in \mathcal{A}\}$. Each logical edge $X(e)$ is assigned the same weight as e .

Then the procedure constructs a collection $\hat{\mathcal{F}}$ of superclusters and a corresponding collection $\hat{\mathcal{T}}$ of spanning subtrees on this logical graph. The construction operates as follows. The procedure first initializes the output partition as $\mathcal{F} = \{\{F\} \mid F \in \mathcal{F}^{k-1}\}$; i.e., each vertex of $\hat{V} = \mathcal{F}^{k-1}$ is a separate supercluster. The output collection of spanning subtrees is initialized to $\hat{\mathcal{T}} = \emptyset$. The procedure then inspects the edges of \hat{E} sequentially in increasing order of weight. An inspected logical edge $X(e)$ is added to $\hat{\mathcal{T}}$ if it does not close a cycle with edges already in $\hat{\mathcal{T}}$. Whenever an edge $X(e) = (F_1, F_2)$ is added to $\hat{\mathcal{T}}$, the superclusters \hat{F}_1 and \hat{F}_2 containing F_1 and F_2 , respectively, are merged into one supercluster \hat{F} , setting $\hat{F} = \hat{F}_1 \cup \hat{F}_2$ and eliminating \hat{F}_1 and \hat{F}_2 , and the corresponding spanning subtrees are fused together into a spanning subtree for the new supercluster \hat{F} , setting $\hat{T}(\hat{F}) = \hat{T}(\hat{F}_1) \cup \hat{T}(\hat{F}_2) \cup \{X(e)\}$.

In each step during this process, whenever a logical edge $X(e) = (F_1, F_2)$ between two superclusters \hat{F}_1 and \hat{F}_2 such that $F_1 \in \hat{F}_1$ and $F_2 \in \hat{F}_2$ is inspected, the procedure also considers declaring one or two superclusters *finished* as follows:

- If the step resulted in a merge operation creating a new supercluster $\hat{F} = \hat{F}_1 \cup \hat{F}_2$, then the newly constructed supercluster \hat{F} is declared finished if one of the following conditions hold:
 - e is the heaviest edge in $\mathcal{A}(F_1)$ or in $\mathcal{A}(F_2)$, or
 - either \hat{F}_1 or \hat{F}_2 is finished.
- If the step did not result in a merge between \hat{F}_1 and \hat{F}_2 , then
 - the supercluster \hat{F}_1 is declared finished if e is the heaviest edge in $\mathcal{A}(F_1)$;
 - the supercluster \hat{F}_2 is declared finished if e is the heaviest edge in $\mathcal{A}(F_2)$.

Also, after every edge inspection step, some of the remaining edges become “dangerous” and are removed from the set \mathcal{A} . A remaining logical edge $X(e) = (F_1, F_2)$, $F_1 \in \hat{F}_1$, $F_2 \in \hat{F}_2$, is still “safe” (i.e., not dangerous) if $e \in \mathcal{A}(F_1)$ and the supercluster \hat{F}_1 is still unfinished, or if $e \in \mathcal{A}(F_2)$ and the supercluster \hat{F}_2 is still unfinished. Thus after every edge inspection step, the procedure examines every edge and removes each dangerous edge e from the set \mathcal{A} . The procedure also removes the corresponding logical edge $X(e)$ from \hat{E} . The process terminates once all superclusters are declared finished (which, as can easily be verified, happens concurrently with the set \mathcal{A} becoming empty).

In the second stage, the procedure transforms the superclusters and spanning subtrees constructed for \hat{G} into ones for the original graph G . Specifically, for every supercluster $\hat{F} \in \hat{\mathcal{F}}$ of the logical graph \hat{G} , with spanning subtree $\hat{T}(\hat{F})$, the procedure merges the original clusters included in the supercluster \hat{F} into a cluster F' of G and creates the corresponding spanning subtree $T(F')$ for this cluster by merging $\hat{T}(\hat{F})$ together with all the spanning subtrees from the collection \mathcal{T}^{k-1} spanning the original

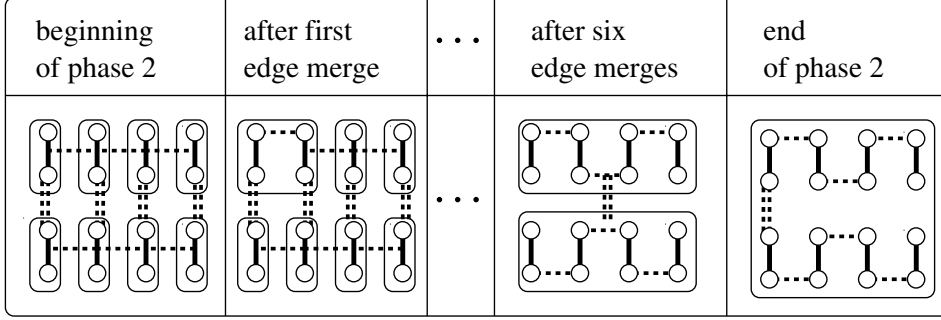


FIG. 3. The operation of Procedure `Const_Frags` during phase 2 on K_{16} and the logical graph structure after various stages of its execution.

clusters included in the supercluster \hat{F} , i.e., setting

$$T(F') = \{e \mid X(e) \in \hat{T}(\hat{F})\} \cup \bigcup_{F \in \hat{F}} T(F).$$

It then adds the cluster F' to the output cluster collection \mathcal{F}^k and the spanning subtree $T(F')$ for it into \mathcal{T}^k .

The operation of Procedure `Const_Frags` during the second phase of the algorithm's execution on our example graph K_{16} is illustrated in Figure 3.

3. Analysis. In this section we prove that the algorithm described in section 2 is correct and analyze its complexity. It is more convenient to start with the complexity analysis.

3.1. Complexity. The following lemma is the key to the complexity analysis. It bounds from below the growth rate of fragments.

Consider phase k of the algorithm. Let \hat{G} be the logical graph constructed by Procedure `Const_Frags`. Let $\hat{\mathcal{F}}$ be the collection of clusters constructed by Procedure `Const_Frags` for \hat{G} . Define μ to be the minimum between the smallest cluster size and number of clusters minus one (cf. line 2 in Procedure `Cheap_Out`).

LEMMA 3.1. *Every supercluster in $\hat{\mathcal{F}}$ consists of at least $\mu + 1$ logical vertices of \hat{G} .*

Proof. To establish the lemma, we prove the following stronger claim: whenever the procedure declares a supercluster \hat{F} finished, it contains at least $\mu + 1$ logical vertices of \hat{G} . This claim is proved by structural induction on the superclusters.

There are three base cases. The first is when \hat{F} is declared finished following a merge step $\hat{F} = \hat{F}_1 \cup \hat{F}_2$ where the two merged superclusters were unfinished. This merge step was based on the inspection of some logical edge $X(e) = (F_1, F_2)$ such that $F_1 \in \hat{F}_1$ and $F_2 \in \hat{F}_2$. By the specification of Procedure `Const_Frags`, without loss of generality we may assume that e is the heaviest edge in $\mathcal{A}(F_1)$. As the edges are inspected in increasing weight order, all other edges in $\mathcal{A}(F_1)$ have already been inspected. There are μ such edges, $e_{i_1}, \dots, e_{i_\mu}$, leading to *distinct* original clusters $F_{j_1}, \dots, F_{j_\mu}$. Whenever an edge e_{i_l} was inspected, either the superclusters containing F_1 and F_{j_l} were merged, or e_{i_l} was found to close a cycle, indicating that F_1 and F_{j_l} already belonged to the same supercluster. Hence the finished supercluster \hat{F} contains (at least) the $\mu + 1$ original clusters $F_1, F_{j_1}, \dots, F_{j_\mu}$.

The second base case is when \hat{F} is declared finished following the inspection of some logical edge $X(e) = (F, F_2)$ such that $F \in \hat{F}$ and $F_2 \in \hat{F}_2$, which did not result in a merge. This happens since e is the heaviest edge in $\mathcal{A}(F)$. Again, all $\mu - 1$ other edges in $\mathcal{A}(F)$ have already been inspected, and by a similar reasoning as above, the finished supercluster \hat{F} contains (at least) $\mu + 1$ original clusters. The third base case is the dual case where \hat{F} is declared finished following the inspection of some logical edge $X(e) = (F_1, F)$ such that $F_1 \in \hat{F}_1$ and $F \in \hat{F}$, which did not result in a merge. Again this happens since e is the heaviest edge in $\mathcal{A}(F)$, and the claim follows in the same way.

The inductive claim concerns the case where \hat{F} is declared finished following a merge step $\hat{F} = \hat{F}_1 \cup \hat{F}_2$ where one or both of the two merged superclusters were finished. In this case, the claim follows directly from the inductive hypothesis. \square

LEMMA 3.2. *For any cluster $F \in \mathcal{F}^k$, $|F| \geq 2^{2^{k-1}}$.*

Proof. Denote by μ_k the minimum size of a cluster $F \in \mathcal{F}^k$. First, note that for all $k \geq 0$,

$$(3.1) \quad \mu_{k+1} \geq \mu_k(\mu_k + 1).$$

Equation (3.1) is true by Lemma 3.1, which implies that clusters generated in phase $k+1$ consist of the union of at least $\mu_k + 1$ clusters of phase k , each containing at least μ_k nodes. Now, since $\mu_0 = 1$, we have $\mu_1 \geq 2$. Since (3.1) implies that $\mu_{k+1} > \mu_k^2$, we conclude that $\mu_k > \mu_1^{2^{k-1}} = 2^{2^{k-1}}$. \square

COROLLARY 3.3. *The algorithm terminates after at most $\log \log n + 1$ phases.*

Proof. The proof follows from Lemma 3.2, since the algorithm terminates at phase k in which $|F| \geq n$ for any $F \in \mathcal{F}^k$. \square

The following statement is immediate from the code of the algorithm.

LEMMA 3.4. *Each phase requires $O(1)$ rounds.*

We now conclude with the following result.

THEOREM 3.5. *The time complexity of the algorithm is $O(\log \log n)$ rounds, and the overall number of bits communicated is $O(n^2 \log n)$.*

Proof. The time complexity bound follows directly from Corollary 3.3 and Lemma 3.4. For the total number of bits communicated, we account for each step separately as follows. In step 1 of the main algorithm, each node in a cluster F sends messages to all other clusters; i.e., each node sends $|\mathcal{F}^{k-1}| - 1$ messages. Since each cluster is of size at least $2^{2^{k-1}}$ by Lemma 3.2, it follows that $|\mathcal{F}^{k-1}| \leq n/2^{2^{k-1}}$. Therefore, the number of messages sent by a node at step 1 of phase k is less than $n/2^{2^{k-1}}$. Since each message contains at most $c \log n$ bits for some constant c , the number of bits sent over all phases in step 1 is less than

$$\sum_{k=0}^{\log \log n + 1} n \cdot c \log n \cdot \frac{n}{2^{2^{k-1}}} = n^2 c \log n \sum_{k=0}^{\log \log n + 1} 2^{-2^{k-1}} = O(n^2 \log n).$$

No messages are sent in step 2. The number of messages sent in step 3 of the algorithm in each phase is $O(n)$ over all nodes (since each node receives at most one message), for a total of $O(n \log n \log \log n)$ bits throughout the execution. To account for the number of messages sent in steps 4 and 5, we bound the total number of messages sent in that step over all nodes and over all phases: note that each edge added to the MST contributes $O(n \log n)$ bits sent at steps 4 and 5, and since exactly $n - 1$ edges are added to the MST overall, the total number of bits sent in these steps throughout the execution of the algorithm is $O(n^2 \log n)$. The result follows. \square

We note that by the results of Adler et al. [1] applied to our model, the minimal number of bits required to solve the MST problem is $\Omega(n^2 \log n)$ in the worst case.

3.2. Correctness. The correctness of the algorithm is proved by the following invariant.

LEMMA 3.6. *In each phase k , for every cluster $F \in \mathcal{F}^k$ constructed by Procedure Const.Frags, the corresponding spanning tree is a fragment, namely, $T(F) = \mathcal{T}(F)$.*

Proof. The proof is by induction on k . The initial partition, \mathcal{F}^0 , trivially satisfies the claim. Now suppose that the collection \mathcal{T}^{k-1} consists of only MST edges, and consider the collection \mathcal{T}^k constructed in phase k . The spanning subtrees in this collection are composed of spanning subtrees from \mathcal{T}^{k-1} fused together by new edges added by Procedure Const.Frags. It suffices to show that every edge added to the trees of \mathcal{T}^k in phase k is indeed an MST edge. For this, we rely on the standard MST construction rule which says that if e is the lightest outgoing edge incident on a fragment, then it belongs to the MST. Consequently, we have to show that whenever Procedure Const.Frags selects a logical edge $X(e) = (F_1, F_2)$, $F_1 \in \hat{F}_1$, $F_2 \in \hat{F}_2$, and uses it to merge the two superclusters \hat{F}_1 and \hat{F}_2 in G , then e is the lightest edge outgoing from one of the two corresponding clusters $H_1 = \bigcup_{F \in \hat{F}_1} F$ and $H_2 = \bigcup_{F \in \hat{F}_2} F$ in G .

As the edge e has not been erased prior to this step, necessarily either $e \in \mathcal{A}(F_1)$ and \hat{F}_1 is unfinished, or $e \in \mathcal{A}(F_2)$ and \hat{F}_2 is unfinished. Without loss of generality suppose the former. We claim that in this case, e is the lightest outgoing edge incident on H_1 .

Consider some other outgoing edge e' incident on H_1 ; i.e., e' is incident on some fragment $F' \in \hat{F}_1$. Suppose, towards contradiction, that $\omega(e') < \omega(e)$. If $e' \in \mathcal{A}(F')$, then e' should have been considered by Const.Frags before e , and subsequently either added to the spanning subtree $\hat{T}(\hat{F}_1)$ or discarded as an internal edge, in either case contradicting our assumption that e' is an outgoing edge of H_1 (hence $X(e')$ is an outgoing edge of \hat{F}_1). It follows that $e' \notin \mathcal{A}(F')$. Let $X(e') = (F', F'')$. There may be two reasons why e' was not added to $\mathcal{A}(F')$. The first is that some other edge e'' with $X(e'') = (F', F'')$ was already included in $\mathcal{A}(F')$ before e' . In that case, $\omega(e'') < \omega(e')$, and hence also $\omega(e'') < \omega(e)$. This implies that e'' has already been inspected by the procedure at some earlier step. But then the clusters F' and F'' must already belong to the supercluster \hat{F}_1 ; and hence in \hat{F}_1 , the edge e' is internal, a contradiction. The other possible reason why e' was not added to $\mathcal{A}(F')$ is that there are μ lighter edges incident on F' , which were added to $\mathcal{A}(F')$. Letting e'' be the heaviest edge in $\mathcal{A}(F')$ in this case, it follows that $\omega(e'') < \omega(e')$, and hence $\omega(e'') < \omega(e)$. This means that e'' has already been inspected by the procedure at some earlier step. But then the supercluster \hat{F}_0 that contained F' at the end of that step should have been declared finished upon inspection of its heaviest edge. This would necessitate that \hat{F}_1 is finished now, a contradiction.

THEOREM 3.7. *The tree produced by the algorithm is an MST of the graph.*

Proof. The proof follows from Lemma 3.6 and the fact that by Lemma 3.2, \mathcal{F}^k contains exactly one cluster for $k > \log \log n$. \square

4. Extension to larger messages. In this section we extend the algorithm to a model in which each message can contain any number of bits (so long as it is at least $\log n$). Specifically, we assume that each message may contain $\ell \log n$ bits. The extension of the algorithm to this case is straightforward. It turns out that the asymptotic worst-case number of rounds drops to a constant when the message size

is n^ϵ for $\epsilon > 0$, but $\Theta(\log \log n)$ rounds are required by our algorithm for any polylog message size.

First, we explain how to modify the algorithm to use messages that can contain ℓ edges. The idea is to change steps 2(b) (which is the invocation of Procedure `Cheap_Out`) and 3 in the main algorithm so that each node can be the guardian of ℓ edges. Specifically, the modified algorithm is identical to the algorithm of section 2, except for the following steps.

2b*. Perform (locally) Procedure `Cheap_Out*`. This procedure (described below) does the following:

- It selects a set $\mathcal{A}(F)$ containing the $\ell \cdot N$ cheapest edges that go out of F to $\ell \cdot N$ distinct clusters.
- It appoints for each such edge e a *guardian* node $g(e)$ in F , ensuring that each node in F is appointed as guardian to at most ℓ edges.

3*. Let $\{e'_1, \dots, e'_\ell\} \subseteq \mathcal{A}(F)$ be the edges for which v was appointed as guardian, i.e., all edges e'_i such that $g(e'_i) = v$. Send $\{e'_1, \dots, e'_\ell\}$ to v^* , the node with the minimal ID in the graph.

(At the end of this step, v^* knows all the edges in the set $\mathcal{A} = \bigcup_{F' \in \mathcal{F}^{k-1}} \mathcal{A}(F')$.)

The modified `Cheap_Out*` procedure is identical to Procedure `Cheap_Out`, except for the following two steps:

2*. Let $\mu = \min\{\ell \cdot N, |\mathcal{F}^{k-1}| - 1\}$.

5*. Appoint the i th node of F as the guardian of the j th edge added to $\mathcal{A}(F)$ if $j \bmod (\ell \cdot N) = i$.

The correctness of the modification is obvious, as Lemma 3.6 is stated in terms of a general μ , and it relies only on the assumption that $\mathcal{A}(F)$ contains the μ lightest edges connecting F to μ distinct clusters.

The complexity analysis of the generalized algorithm requires a little work. First, we observe that Lemma 3.1 holds without change: it is also stated in terms of a general μ . Lemma 3.4 also holds by the assumption that each message can contain ℓ edges, and since each node is the guardian of at most ℓ messages by the modified procedure `Cheap_Out*`. However, Lemma 3.2 holds only for $\ell = \Theta(1)$. Below we generalize Theorem 3.5 to different values of ℓ .

THEOREM 4.1. *The extended algorithm terminates in $O(\log(\frac{\log n}{\log \ell}))$ rounds, and the total number of bits communicated is $\Theta(n^2 \log n)$.*

Proof. Let μ_k be the smallest possible cluster size after the k th round. By definition, $\mu_0 = 1$. If each guardian node sends ℓ edges, then each cluster merges with at least $\ell \mu_k$ other clusters in the k th phase. It follows that $\mu_{k+1} \geq (\ell \mu_k + 1) \mu_k > \ell \mu_k^2$. Since $\mu_0 = 1$, we have $\mu_1 > \ell$, and therefore $\mu_k > \ell^{2^{k-1}}$. The bound on the number of rounds follows, since the algorithm terminates when $\mu_k \geq n$, and each phase takes only $O(1)$ rounds by Lemma 3.4. For the total number of bits communicated by the algorithm, we observe that the only difference is the number of messages sent in step 3*. Let B_3 denote the total number of bits sent in step 3* throughout the execution of the algorithm. We claim that $B_3 = O(n^2 \log n)$. To see this, note that the total number of edge identifiers sent in step 3 in a single phase, over all nodes, is $O(n\ell)$. It follows that $B_3 = O(Tn\ell \log n)$, where T is the number of phases. As shown

above, $T = O(\log(\frac{\log n}{\log \ell}))$. This means that $B_3 = O(n^2 \log n)$: If $1 < \ell < n/\log \log n$, then T is bounded by $O(\log \log n)$; and if $\ell \geq n/\log \log n$, then $T = O(1)$. \square

Let us interpret the result of Theorem 4.1 in two typical cases. First, if ℓ is polynomial in n , i.e., $\ell = n^\epsilon$ for some $\epsilon > 0$, then the total running time of the algorithm is $O(\log(1/\epsilon))$. However, the number of rounds remains $\Theta(\log \log n)$ if ℓ is only polylogarithmic in n . (In fact, it remains $O(\log \log n)$ even if ℓ is as large as $(\log n)^{(\log n)^{1-\epsilon}}$ for some constant $\epsilon > 0$.)

5. Conclusion. This paper shows that an MST can be constructed in sublogarithmic time, even if each message can contain only a constant number of edges. We believe that the algorithm may be useful in some overlay networks. Theoretically, important gaps remain. While there are nontrivial lower bounds on the running time of MST construction in graphs of diameter 3 or more, currently no superconstant lower bound is known even for graphs of diameter 2. We do not know whether there exist lower bounds or better algorithms for graphs of diameter 1 and 2 (recall that the fastest known algorithm for diameter 2 runs in $O(\log n)$ rounds).

Acknowledgment. We thank the anonymous reviewers, whose comments helped improve the presentation of the paper.

REFERENCES

- [1] M. ADLER, W. DITTRICH, B. JUURLINK, M. KUTYLÓWSKI, AND I. RIEPING, *Communication-optimal parallel minimum spanning tree algorithms*, in Proceedings of the 1998 ACM Symposium on Parallel Algorithms and Architecture, 1998, pp. 27–36.
- [2] B. AWERBUCH, *Complexity of network synchronization*, J. ACM, 32 (1985), pp. 804–823.
- [3] B. AWERBUCH, *Optimal distributed algorithms for minimum weight spanning tree, counting, leader election and related problems*, in Proceedings of the 19th Annual ACM Symposium on Theory of Computing, 1987, pp. 230–240.
- [4] B. AWERBUCH AND Y. SHILOACH, *New connectivity and MSF algorithms for the shuffle-exchange network and PRAM*, IEEE Trans. Comput., 36 (1987), pp. 1258–1263.
- [5] R. COLE, P. N. KLEIN, AND R. E. TRAJAN, *Finding minimum spanning forests in logarithmic time and linear work using random sampling*, in Proceedings of the ACM Symposium on Parallel Algorithms and Architecture, 1996, pp. 243–250.
- [6] R. G. GALLAGER, P. A. HUMBLET, AND P. M. SPIRA, *A distributed algorithm for minimum-weight spanning trees*, ACM Trans. Prog. Lang. and Syst., 5 (1983), pp. 66–77.
- [7] J. A. GARAY, S. KUTTEN, AND D. PELEG, *A sublinear time distributed algorithm for minimum-weight spanning trees*, SIAM J. Comput., 27 (1998), pp. 302–316.
- [8] J. JANNOTTI, D. GIFFORD, K. L. JOHNSON, M. F. KAASHOEK, AND J. J. W. O’TOOLE, *Overcast: Reliable multicasting with an overlay network*, in Proceedings of the 4th Annual USENIX OSDI, 2000, pp. 197–212.
- [9] D. R. KARGER, P. N. KLEIN, AND R. E. TRAJAN, *A randomized linear time algorithm to find minimum spanning trees*, J. ACM, 42 (1995), pp. 321–328.
- [10] Z. LOTKER, B. PATT-SHAMIR, AND D. PELEG, *Distributed MST for constant diameter graphs*, in Proceedings of the 20th Annual ACM Symposium on Principles of Distributed Computing, 2001, pp. 63–71.
- [11] N. LYNCH, *Distributed Algorithms*, Morgan Kaufmann, San Mateo, CA, 1995.
- [12] D. PELEG AND V. RUBINOVICH, *Near-tight lower bound on the time complexity of distributed minimum-weight spanning tree construction*, SIAM J. Comput., 30 (2000), pp. 1427–1442.
- [13] J. H. SALTZER, D. P. REED, AND D. D. CLARK, *End-to-end arguments in system design*, ACM Transactions on Computer Systems, 2 (1984), pp. 277–288.
- [14] I. STOICA, R. MORRIS, D. KARGER, M. F. KAASHOEK, AND H. BALAKRISHNAN, *Chord: A scalable peer-to-peer lookup service for internet applications*, in Proceedings of the ACM SIGCOMM ’01 Conference, San Diego, CA, 2001.
- [15] R. E. TRAJAN, *Data Structures and Network Algorithms*, CBMS-NSF Regional Conf. Ser. in Appl. Math. 44, SIAM, Philadelphia, 1983.

THE COMPLEXITY OF APPROXIMATING THE ENTROPY*

TUĞKAN BATU[†], SANJOY DASGUPTA[‡], RAVI KUMAR[§], AND RONITT RUBINFELD[¶]

Abstract. We consider the problem of approximating the entropy of a discrete distribution under several different models of oracle access to the distribution. In the evaluation oracle model, the algorithm is given access to the explicit array of probabilities specifying the distribution. In this model, linear time in the size of the domain is both necessary and sufficient for approximating the entropy.

In the generation oracle model, the algorithm has access only to independent samples from the distribution. In this case, we show that a γ -multiplicative approximation to the entropy can be obtained in $O(n^{(1+\eta)/\gamma^2} \log n)$ time for distributions with entropy $\Omega(\gamma/\eta)$, where n is the size of the domain of the distribution and η is an arbitrarily small positive constant. We show that this model does not permit a multiplicative approximation to the entropy in general. For the class of distributions to which our upper bound applies, we obtain a lower bound of $\Omega(n^{1/(2\gamma^2)})$.

We next consider a combined oracle model in which the algorithm has access to both the generation and the evaluation oracles of the distribution. In this model, significantly greater efficiency can be achieved: we present an algorithm for γ -multiplicative approximation to the entropy that runs in $O((\gamma^2 \log^2 n)/(h^2(\gamma - 1)^2))$ time for distributions with entropy $\Omega(h)$; for such distributions, we also show a lower bound of $\Omega((\log n)/(h(\gamma^2 - 1) + \gamma^2))$.

Finally, we consider two special families of distributions: those in which the probabilities of the elements decrease monotonically with respect to a known ordering of the domain, and those that are uniform over a subset of the domain. In each case, we give more efficient algorithms for approximating the entropy.

Key words. entropy estimation, sublinear algorithms, properties of distributions, property testing

AMS subject classifications. 94A17, 94A20, 68W25, 62G99

DOI. 10.1137/S0097539702403645

1. Introduction. The Shannon entropy is a measure of the randomness of a distribution and plays a central role in statistics, information theory, and coding theory. The entropy of a random source sheds light on the inherent compressibility of data produced by such a source. In this paper we consider the complexity of approximating the entropy under various assumptions on the way the input is presented.

Suppose the algorithm has access to an *evaluation oracle*¹ in which the distribution \mathbf{p} is given as an array whose i th location contains the probability p_i assigned to the i th element of the domain. It is clear that an algorithm that reads the entire representation can calculate the exact entropy. However, it is also easy to see that in this model, time linear in the size of the domain is required even to approximate the entropy: consider two distributions, one with a singleton support set (zero entropy) and the other with a two-element support set (positive entropy). Any algorithm that

*Received by the editors March 6, 2002; accepted for publication (in revised form) April 21, 2005; published electronically October 3, 2005. A preliminary version of this paper appeared in the *Proceedings of the 34th Annual ACM Symposium on Theory of Computing*, 2002, pp. 678–687.

<http://www.siam.org/journals/sicomp/35-1/40364.html>

[†]School of Computing Science, Simon Fraser University, Burnaby, BC, Canada V5A 1S6 (batu@cs.sfu.ca). This author's work was supported by ONR N00014-97-1-0505, MURI.

[‡]Department of Computer Science and Engineering, University of California, San Diego, CA 92093 (dasgupta@cs.ucsd.edu).

[§]IBM Almaden Research Center, San Jose, CA 95120 (ravi@almaden.ibm.com).

[¶]CSAIL, MIT, Cambridge, MA 02139 (ronitt@csail.mit.edu).

¹We use the terminology from [9].

approximates the entropy to within a multiplicative factor must distinguish between these two distributions, and a randomized algorithm for distinguishing two such distributions requires linear time in general.

Next suppose the distribution $\mathbf{p} = \langle p_1, \dots, p_n \rangle$ is given as a *generation oracle* that draws samples from it. This model has been considered in the statistics, physics, and information theory communities (cf. [1, 4, 7, 8, 10, 11, 12, 14, 15, 16]). None of these previous works, however, provides a rigorous analysis of computational efficiency and sample complexity in terms of the approximation quality and the domain size. To the best of our knowledge, the only previously known algorithms that do not require superlinear (in the domain size) sample complexity are those presented in [10, 14, 12], which give good estimates in some cases, such as when the infinity norm of the distribution is small. Some of these algorithms use an estimate of the collision probability, $\|\mathbf{p}\|^2$, to give a lower bound estimate of the entropy: using Jensen's inequality, it is shown [14] that

$$\log \|\mathbf{p}\|^2 = \log \sum_i p_i^2 \geq \sum_i p_i \log p_i = -H(\mathbf{p}).$$

In fact, when the infinity norm $\|\mathbf{p}\|_\infty$ of \mathbf{p} is at most $n^{-\alpha}$ (in other words, when the min-entropy of \mathbf{p} is large), the collision probability can be used to give an upper bound on the entropy of the distribution: using the relationship between norms,

$$\log \|\mathbf{p}\|^2 \leq \log(\|\mathbf{p}\|_\infty) \leq \log n^{-\alpha} = -\alpha \log n \leq -\alpha \cdot H(\mathbf{p}).$$

It is, however, unclear how to use the collision probability to obtain an arbitrary multiplicative approximation with a better sample complexity than our results (stated below), since approximating the collision probability itself will require $\Omega(\sqrt{n})$ samples. However, the collision probability can be used to understand much more about certain types of distributions; for instance, it exactly determines the entropy in the special case of distributions that are known to be uniform over an unknown subset of arbitrary size (see section 7).

1.1. Our results. In this section, we summarize the results in this paper. Table 1 also presents some of the upper and lower bounds proved in the paper.

(1) THE GENERATION ORACLE MODEL. When the distribution is given as a generation oracle, we show that the entropy can be approximated well in sublinear time for a large class of distributions. Informally, a γ -multiplicative approximation to the entropy can be obtained in time $O(n^{(1+\eta)/\gamma^2} \log n)$, where n is the size of the domain of the distribution and η is an arbitrarily small positive constant, provided that the distribution has $\Omega(\gamma/\eta)$ entropy. Our algorithm is simple—we partition the elements in the domain into big or small elements based on their probability masses and approximate the entropy of the big and small elements separately by different methods. On the other hand, we show that one cannot obtain a multiplicative approximation to the entropy in general. Furthermore, even for the class of distributions to which our upper bound applies, we obtain a lower bound of $\Omega(n^{1/(2\gamma^2)})$. In the conference version of this paper, we also claimed another lower bound of $\Omega(n^{2/(5\gamma^2-2)})$ for this problem. We retract this bound, as there was a gap in our proof.

It is interesting to consider what these bounds imply for the complexity of achieving a 2-approximation for distributions with nonconstant entropy. Our upper bound yields an algorithm that runs in $\tilde{O}(n^{(1+o(1))/4})$ time, while our lower bound demonstrates that a running time of at least $\Omega(n^{1/8})$ is necessary.

(2) **THE EVALUATION ORACLE MODEL.** When the distribution is given as an evaluation oracle, we show a lower bound of $\Omega(n2^{-\gamma^2(h+1)})$ on the number of oracle accesses needed to γ -approximate the entropy for the class of distributions with entropy at least h .

(3) **THE COMBINED ORACLE MODEL.** We then consider a *combined oracle* model, in which the algorithm has access to both the generation and the evaluation oracles of the distribution. We assume that the two oracles are consistent, which is a natural assumption for such a model. In the combined oracle model, we give a γ -approximation algorithm that runs in time $O((\gamma^2 \log^2 n)/(h^2(\gamma-1)^2))$ for distributions with entropy $\Omega(h)$; we also show a lower bound of $\Omega((\log n)/(h(\gamma^2-1)+\gamma^2))$ for this class of distributions. For example, to achieve a constant approximation for distributions with entropy $\Omega(h)$, our algorithm runs in time $O((1/h^2) \log^2 n)$, while our lower bound is $\Omega((1/h) \log n)$, that is, quadratically smaller than the upper bound.

(4) **SPECIAL FAMILIES OF DISTRIBUTIONS.** Finally, we consider two families of distributions for which we show more efficient upper bounds. The first family is that of *monotone distributions*, in which the probabilities decrease monotonically over some known ordering of the elements (i.e., $p_i \geq p_{i+1}$). We give an $O((1 + \log^{-1} \gamma) \log n)$ -time (resp., $O((\log n)^6 \text{poly}(\gamma))$ -time) algorithm for γ -approximating the entropy in the evaluation oracle model (resp., generation oracle model). The second family is that of *subset-uniform distributions*, in which the distribution is uniform over some subset of the domain. In this case we give $O(\sqrt{k})$ -time algorithms for approximating the entropy, where k is the size of the support set.

TABLE 1
Our results for γ -approximation, where $\gamma > 1$.

Model		Lower bound	Upper bound
Evaluation oracle:	General	$\Omega(n)$	$O(n)$
	$H(\mathbf{p}) \geq h$	$\Omega(n2^{-\gamma^2(h+1)})$, Thm. 8	?
Generation oracle:	General	∞ , Thm. 6	—
	High enough entropy	$\Omega(n^{1/(2\gamma^2)})$, Thm. 7, $H(\mathbf{p}) > \Omega((\log n)/\gamma^2)$	$\tilde{O}(n^{1/\gamma^2})$, Thm. 2, $H(\mathbf{p}) > \Omega(\gamma)$
Combined oracle:	General	$\Omega(n^{(1-o(1))/\gamma^2})$, Thm. 12	?
	$H(\mathbf{p}) \geq h$	$\Omega\left(\frac{\log n}{h(\gamma^2-1)+\gamma^2}\right)$, Thm. 13	$O\left(\frac{\gamma^2 \log^2 n}{h^2(\gamma-1)^2}\right)$, Thm. 9

1.2. Related work. The work of Goldreich and Vadhan [6] considers the complexity of approximating the entropy in a different model in which a distribution Y is encoded as a circuit $Y = C(X)$ whose input X is uniformly distributed; in this model, they show that a version of the problem is complete for statistical zero-knowledge. Their version of the problem could be viewed as an additive approximation to the entropy.

The work of [3] and [2] considers algorithms for testing other properties of distributions in the generation oracle model. The properties considered are whether two input distributions are close or far, and whether a joint distribution is independent, respectively. Both papers give algorithms whose sample complexity is sublinear in the domain size, along with lower bounds showing the algorithms to be nearly optimal.

1.3. Organization. In section 2, we introduce the basic definitions used in this paper. In section 3, we give algorithms and lower bounds for the generation oracle model. Section 4 describes a lower bound for the evaluation oracle model, and section 5 gives algorithms and lower bounds for the combined oracle case. Finally, in sections 6 and 7, we give more efficient algorithms for two families of distributions.

2. Preliminaries. We consider discrete distributions over a domain of size n , which we denote by $[n] \stackrel{\text{def}}{=} \{1, \dots, n\}$. Let $\mathbf{p} = \langle p_1, \dots, p_n \rangle$ be such a distribution, where $p_i \geq 0$, $\sum_{i=1}^n p_i = 1$. An algorithm is said to have *evaluation oracle* access to the distribution \mathbf{p} if oracle query i is answered by p_i . An algorithm is said to have *generation oracle* access to \mathbf{p} if it is given a source that draws samples independently from \mathbf{p} . An algorithm has *combined oracle* access to \mathbf{p} if it has both evaluation and generation oracle access to \mathbf{p} . We say the algorithm is in model \mathcal{O} if it has oracle access of type \mathcal{O} to the distribution.

The entropy of distribution \mathbf{p} is defined as

$$H(\mathbf{p}) \stackrel{\text{def}}{=} - \sum_{i=1}^n p_i \log p_i,$$

where all the logarithms are to the base 2. For a set $S \subseteq [n]$, we define $w_{\mathbf{p}}(S) \stackrel{\text{def}}{=} \sum_{i \in S} p_i$, and we define the contribution of S to the entropy as

$$H_S(\mathbf{p}) \stackrel{\text{def}}{=} - \sum_{i \in S} p_i \log p_i.$$

Notice that $H_S(\mathbf{p}) + H_{[n] \setminus S}(\mathbf{p}) = H(\mathbf{p})$.

The L_2 -norm of distribution \mathbf{p} is $\|\mathbf{p}\| \stackrel{\text{def}}{=} \sqrt{\sum_{i=1}^n p_i^2}$ and the L_∞ -norm of \mathbf{p} is $\|\mathbf{p}\|_\infty \stackrel{\text{def}}{=} \max_{i=1}^n p_i$. We denote the L_1 -distance between two distributions \mathbf{p}, \mathbf{q} by $|\mathbf{p} - \mathbf{q}| \stackrel{\text{def}}{=} \sum_{i=1}^n |p_i - q_i|$.

The following lemma summarizes some upper and lower bounds on entropy that will turn out to be useful at many points in the paper.

LEMMA 1. *Pick any $S \subseteq [n]$.*

(a) *The partial entropy $H_S(\mathbf{p})$ is maximized when $w_{\mathbf{p}}(S)$ is spread uniformly over the set $|S|$:*

$$H_S(\mathbf{p}) \leq w_{\mathbf{p}}(S) \cdot \log(|S|/w_{\mathbf{p}}(S)) \leq w_{\mathbf{p}}(S) \cdot \log |S| + (\log e)/e.$$

(b) *Suppose there is some $\beta \leq 1/e$ such that $p_i \leq \beta$ for all $i \in S$. Then $H_S(\mathbf{p}) \leq \beta |S| \log(1/\beta)$.*

(c) *Suppose there is some β such $\beta \leq p_i \leq 1/e$ for all $i \in S$. Then $H_S(\mathbf{p}) \geq \beta |S| \log(1/\beta)$.*

(d) *Suppose there is some β such that $p_i \leq \beta$ for all $i \in S$. Then $H_S(\mathbf{p}) \geq w_{\mathbf{p}}(S) \log(1/\beta)$.*

Proof. Statement (a) follows from the concavity of the logarithm function. By Jensen's inequality,

$$\frac{1}{w_{\mathbf{p}}(S)} \cdot H_S(\mathbf{p}) = \sum_{i \in S} \frac{p_i}{w_{\mathbf{p}}(S)} \log \frac{1}{p_i} \leq \log \left(\sum_{i \in S} \frac{p_i}{w_{\mathbf{p}}(S)} \cdot \frac{1}{p_i} \right) = \log \frac{|S|}{w_{\mathbf{p}}(S)}.$$

Therefore,

$$H_S(\mathbf{p}) \leq w_{\mathbf{p}}(S) \cdot \log \frac{|S|}{w_{\mathbf{p}}(S)} \leq w_{\mathbf{p}}(S) \cdot \log |S| + \frac{\log e}{e}.$$

The last inequality comes from observing that the function $x \log(1/x)$ is zero at $x = 0, 1$ and has a single local maximum in the interval $[0, 1]$ at $x = 1/e$.

Statement (b) follows immediately from the previous observation about $x \log(1/x)$, which implies that, under the given constraint, $H_S(\mathbf{p})$ is maximized by setting all the p_i to β .

The proof of statement (c) also follows from the concavity of $x \log(1/x)$; under the given constraints, $p_i \log(1/p_i)$ is minimized when $p_i = \beta$.

For statement (d), we notice that $H_S(\mathbf{p})$ is strictly concave; therefore, over any closed domain, it is minimized at a boundary point. In particular, when the domain is $[0, \beta]^{|S|}$, the minimum point must have some coordinate with $p_i = 0$ or $p_i = \beta$. We can now restrict our attention to the remaining coordinates and apply the same argument again. In this way, we find that the minimum is realized when $w_{\mathbf{p}}(S)/\beta$ of the p_i are β , and the rest are zero. \square

Let $\gamma > 1$ and let \mathcal{D} be a family of distributions. We say that \mathcal{A} is an algorithm in model \mathcal{O} for γ -approximating the entropy of a distribution in \mathcal{D} if, for every $\mathbf{p} \in \mathcal{D}$, given oracle access of type \mathcal{O} to \mathbf{p} , algorithm \mathcal{A} outputs a value $\mathcal{A}(\mathbf{p})$ such that $H(\mathbf{p})/\gamma \leq \mathcal{A}(\mathbf{p}) \leq \gamma \cdot H(\mathbf{p})$ with probability at least $3/4$. (This probability of success can generically be increased to $1 - \delta$ by running the algorithm $\log(1/\delta)$ times and returning the median of the values.) The time complexity of \mathcal{A} is specified as a function of γ and n . We will use the notation \mathcal{D}_h to denote the family of distributions with entropy at least h .

3. The generation oracle model.

3.1. Upper bounds. In this section we obtain an algorithm for estimating the entropy of a large class of distributions in the generation oracle model. We prove the following theorem.

THEOREM 2. *For every $\gamma > 1$ and every ϵ_o such that $0 < \epsilon_o \leq 1/2$, there exists an algorithm in the generation oracle model that runs in time $O((n^{1/\gamma^2}/\epsilon_o^2) \cdot \log n)$ and, with success probability at least $3/4$, returns a $(1 + 2\epsilon_o)\gamma$ -approximation to the entropy of any distribution on $[n]$ in $\mathcal{D}_{4\gamma/(\epsilon_o(1-2\epsilon_o))}$.*

Given any $\eta > 0$ and $\gamma' > 1$, one can set $\gamma = \gamma'/(1+2\epsilon_o)$ above and choose ϵ_o small enough to yield a γ' -approximation algorithm with running time $O(n^{(1+\eta)/\gamma'^2} \log n)$ for distributions of entropy $\Omega(\gamma/\eta)$. Note that choosing η to be small affects both the running time and the family of distributions to which the algorithm can be applied.

The main idea behind the algorithm is the following. We classify elements in $[n]$ as either big or small, depending on their probability mass. For a fixed $\alpha > 0$ and a distribution \mathbf{p} , the set of indices with high probabilities (the *big* elements) is defined as

$$B_\alpha(\mathbf{p}) \stackrel{\text{def}}{=} \{i \in [n] \mid p_i \geq n^{-\alpha}\}.$$

We then approximate the contributions of the big and small elements to the entropy separately. Section 3.1.1 shows how to approximate the entropy of the big elements, section 3.1.2 shows how to approximate the entropy of the small elements, and section 3.1.3 combines these approximations to yield Theorem 2.

3.1.1. Approximating the entropy of the big elements. To estimate the amount by which the big elements contribute to the entropy, we approximate each of their probabilities by drawing samples from the generation oracle.

LEMMA 3. *For every α such that $0 < \alpha \leq 1$, every ϵ_o such that $0 < \epsilon_o \leq 1/2$, and sufficiently large n , there is an algorithm that uses $O((n^\alpha/\epsilon_o^2) \cdot \log n)$ samples from \mathbf{p} and outputs a distribution \mathbf{q} over $[n]$ such that with probability at least $1 - n^{-1}$, the following hold for all i :*

1. *If $p_i \geq \frac{1-\epsilon_o}{1+\epsilon_o}n^{-\alpha}$ (in particular, if $i \in B_\alpha(\mathbf{p})$), then $|p_i - q_i| \leq \epsilon_o p_i$, and*
2. *if $p_i \leq \frac{1-\epsilon_o}{1+\epsilon_o}n^{-\alpha}$, then $q_i \leq (1 - \epsilon_o)n^{-\alpha}$.*

Proof. Let $m = (18n^\alpha/\epsilon_o^2) \cdot \log 2n$. Fix any $i \in [n]$ and define X_j to be the indicator variable that the j th sample is i . Let $q_i = \sum X_j/m$, the average of independent, identically distributed Boolean random variables. If $p_i \geq \frac{1-\epsilon_o}{1+\epsilon_o}n^{-\alpha}$, then by Chernoff bounds,

$$\Pr[|p_i - q_i| > \epsilon_o p_i] \leq 2 \exp\left(-\frac{\epsilon_o^2 p_i m}{3}\right) \leq \frac{1}{2n^2}.$$

Moving onto smaller elements, we again can use Chernoff bounds to show that if $p_i < \frac{1-\epsilon_o}{1+\epsilon_o}n^{-\alpha}$, then

$$\begin{aligned} \Pr[q_i > (1 - \epsilon_o)n^{-\alpha}] &= \Pr[q_i - p_i > (1 - \epsilon_o)n^{-\alpha} - p_i] \\ &\leq \Pr\left[q_i - p_i > \frac{\epsilon_o(1 - \epsilon_o)}{1 + \epsilon_o}n^{-\alpha}\right] \\ &\leq \exp\left(-\left(\frac{\epsilon_o(1 - \epsilon_o)n^{-\alpha}}{(1 + \epsilon_o)p_i}\right)^2 \cdot \frac{p_i m}{3}\right) \\ &\leq \exp\left(-\left(\frac{1 - \epsilon_o}{1 + \epsilon_o}\right)^2 \cdot \frac{6n^{-\alpha}}{p_i} \cdot \log 2n\right) \\ &\leq \exp(-2 \log 2n) \leq \frac{1}{2n^2}. \end{aligned}$$

Statements (1) and (2) of the lemma follow from a union bound over all i . \square

The following lemma shows that the contribution of the big elements $B_\alpha(\mathbf{p})$ to the entropy can be approximated well using \mathbf{q} instead of \mathbf{p} .

LEMMA 4. *Pick any $B \subseteq [n]$. Let $\epsilon_o \in (0, 1)$ be chosen so that each $i \in B$ satisfies $|p_i - q_i| \leq \epsilon_o p_i$. Then,*

$$|H_B(\mathbf{q}) - H_B(\mathbf{p})| \leq \epsilon_o \cdot H_B(\mathbf{p}) + 2\epsilon_o \cdot w_{\mathbf{p}}(B).$$

Proof. For $i \in B$, write $q_i = (1 + \varepsilon_i)p_i$. We know that $|\varepsilon_i| \leq \epsilon_o$.

$$\begin{aligned} H_B(\mathbf{q}) - H_B(\mathbf{p}) &= -\sum_{i \in B} (1 + \varepsilon_i)p_i \log((1 + \varepsilon_i)p_i) + \sum_{i \in B} p_i \log p_i \\ &= -\sum_{i \in B} (1 + \varepsilon_i)p_i \log p_i - \sum_{i \in B} (1 + \varepsilon_i)p_i \log(1 + \varepsilon_i) + \sum_{i \in B} p_i \log p_i \\ &= -\sum_{i \in B} \varepsilon_i p_i \log p_i - \sum_{i \in B} (1 + \varepsilon_i)p_i \log(1 + \varepsilon_i). \end{aligned}$$

By the triangle inequality,

$$\begin{aligned} |H_B(\mathbf{q}) - H_B(\mathbf{p})| &\leq \left| -\sum_{i \in B} \varepsilon_i p_i \log p_i \right| + \left| \sum_{i \in B} (1 + \varepsilon_i) p_i \log(1 + \varepsilon_i) \right| \\ &\leq \sum_{i \in B} -|\varepsilon_i| p_i \log p_i + \sum_{i \in B} p_i |(1 + \varepsilon_i) \log(1 + \varepsilon_i)| \\ &\leq \epsilon_o \cdot H_B(\mathbf{p}) + 2\epsilon_o \cdot w_{\mathbf{p}}(B). \end{aligned}$$

The last step above uses that for $|\varepsilon| \leq \epsilon_o \leq 1$, $|(1 + \varepsilon) \log(1 + \varepsilon)| \leq 2|\varepsilon| \leq 2\epsilon_o$. \square

3.1.2. Approximating the entropy of the small elements. We now estimate the entropy contribution of the small elements. Let S be any subset of small elements, that is, $S \subseteq [n] \setminus B_\alpha(\mathbf{p})$.

If $w_{\mathbf{p}}(S) \leq n^{-\alpha}$, then the contribution of S to the entropy is below any constant and can be ignored for approximation purposes. So, we may assume without loss of generality that $w_{\mathbf{p}}(S) \geq n^{-\alpha}$. Let $\hat{w}(S)$ be the empirical estimate of the probability mass of S , in other words, the number of samples from S divided by the total number of samples. The following lemma bounds the accuracy of this estimate.

LEMMA 5. *If $S \subseteq [n]$ satisfies $w_{\mathbf{p}}(S) \geq n^{-\alpha}$ and if $m = O((n^\alpha/\epsilon_o^2) \log n)$ samples are drawn from \mathbf{p} , then with probability at least $1 - n^{-1}$, the empirical estimate $\hat{w}(S)$ satisfies $(1 - \epsilon_o) \cdot w_{\mathbf{p}}(S) \leq \hat{w}(S) \leq (1 + \epsilon_o) \cdot w_{\mathbf{p}}(S)$. Moreover, if $w_{\mathbf{p}}(S) < n^{-\alpha}$, then $\hat{w}(S) < (1 + \epsilon_o)n^{-\alpha}$.*

Proof. Let X_j be the indicator random variable that takes value 1 when the j th sample lies in S , and let $X = \sum X_j$. Then X is $m\hat{w}(S)$ and it has expected value $E[X] = m \cdot w_{\mathbf{p}}(S)$. The lemma follows by Chernoff bounds and the fact that $w_{\mathbf{p}}(S) \geq n^{-\alpha}$. Similar to the proof of Lemma 3, we can show that if $w_{\mathbf{p}}(S) < n^{-\alpha}$, then $\hat{w}(S) < (1 + \epsilon_o)n^{-\alpha}$. \square

Since $p_i < n^{-\alpha}$ for $i \in S$, by Lemma 1(a),(d), we have that

$$\alpha w_{\mathbf{p}}(S) \log n \leq H_S(\mathbf{p}) \leq w_{\mathbf{p}}(S) \log n + (\log e)/e.$$

Hence, using estimate $\hat{w}(S)$ for $w_{\mathbf{p}}(S)$, we get an approximation to $H_S(\mathbf{p})$.

3.1.3. Putting it together. In this section we describe our approximation algorithm for $H(\mathbf{p})$ and prove Theorem 2. The following is our algorithm for obtaining a γ -approximation to the entropy:

ALGORITHM APPROXIMATEENTROPY (γ, ϵ_o) .

1. Set $\alpha = 1/\gamma^2$.
2. Get $m = O((n^\alpha/\epsilon_o^2) \cdot \log n)$ samples from \mathbf{p} .
3. Let \mathbf{q} be the empirical probabilities of the n elements; that is, q_i is the frequency of i in the sample divided by m . Let $B = \{i \mid q_i > (1 - \epsilon_o)n^{-\alpha}\}$.
4. Output $H_B(\mathbf{q}) + \frac{w_{\mathbf{q}}([n] \setminus B) \log n}{\gamma}$.

Notice that the set B is an empirically determined substitute for $B_\alpha(\mathbf{p})$. We now prove that this algorithm satisfies Theorem 2.

Proof of Theorem 2. First of all, Lemma 3 assures us that with probability at least $1 - 1/n$, two conditions hold: (1) $B_\alpha(\mathbf{p}) \subseteq B$, and (2) every element $i \in B$ satisfies $|p_i - q_i| \leq \epsilon_o p_i$. For the rest of the proof, we will assume that these conditions hold.

Let $S = [n] \setminus B$. Assume for the moment that $w_{\mathbf{p}}(S) \geq n^{-\alpha}$. In this case, we know from Lemma 5 that, with high probability, $|w_{\mathbf{q}}(S) - w_{\mathbf{p}}(S)| \leq \epsilon_o w_{\mathbf{p}}(S)$. Lemma 1(a),(d) tells us that

$$\alpha w_{\mathbf{p}}(S) \log n \leq H_S(\mathbf{p}) \leq w_{\mathbf{p}}(S) \log n + (\log e)/e.$$

Then by Lemma 4,

$$\begin{aligned}
H_B(\mathbf{q}) + \frac{w_{\mathbf{q}}(S) \log n}{\gamma} &\leq (1 + \epsilon_o) \cdot H_B(\mathbf{p}) + 2\epsilon_o + \frac{1 + \epsilon_o}{\gamma} \cdot w_{\mathbf{p}}(S) \log n \\
&\leq (1 + \epsilon_o)(H_B(\mathbf{p}) + \gamma \cdot H_S(\mathbf{p})) + 2\epsilon_o \\
&\leq (1 + \epsilon_o)\gamma \cdot H(\mathbf{p}) + 2\epsilon_o \\
&\leq (1 + 2\epsilon_o)\gamma \cdot H(\mathbf{p})
\end{aligned}$$

if $H(\mathbf{p}) \geq 2/\gamma$. Similarly,

$$\begin{aligned}
H_B(\mathbf{q}) + \frac{w_{\mathbf{q}}(S) \log n}{\gamma} &\geq (1 - \epsilon_o) \cdot H_B(\mathbf{p}) - 2\epsilon_o + \frac{1 - \epsilon_o}{\gamma} \cdot w_{\mathbf{p}}(S) \log n \\
&\geq (1 - \epsilon_o) \left(H_B(\mathbf{p}) + \frac{(H_S(\mathbf{p}) - e^{-1} \log e)}{\gamma} \right) - 2\epsilon_o \\
&= (1 - \epsilon_o) \left(H_B(\mathbf{p}) + \frac{H_S(\mathbf{p})}{\gamma} \right) - \frac{1 - \epsilon_o}{\gamma} e^{-1} \log e - 2\epsilon_o \\
&\geq \frac{H(\mathbf{p})}{(1 + 2\epsilon_o)\gamma}
\end{aligned}$$

if $H(\mathbf{p}) \geq \frac{4\gamma}{\epsilon_o(1-2\epsilon_o)} \geq 2/\gamma$.

It remains to handle the case when $w_{\mathbf{p}}(S)$ is less than $n^{-\alpha}$. Lemma 5 tells us that $w_{\mathbf{q}}(S)$ is with high probability at most $(1 + \epsilon_o)n^{-\alpha}$. Therefore, our estimate of the entropy from small elements, $(w_{\mathbf{q}}(S) \log n)/\gamma$, lies somewhere between zero and $((1 + \epsilon_o)n^{-\alpha} \log n)/\gamma$. For any γ bounded away from one, this is only a negligible contribution to $H(\mathbf{p})$, well within the approximation bound. \square

3.2. Lower bounds. In this section we prove lower bounds on the number of samples needed to approximate the entropy of a distribution to within a multiplicative factor of $\gamma > 1$. All of our lower bounds are shown by exhibiting pairs of distributions that have very different entropies, with ratio at least γ^2 , and yet are hard to distinguish given only a few samples.

3.2.1. Impossibility of approximating entropy in general. First we show that there is no algorithm for computing entropy that can guarantee a bounded approximation factor for all input distributions. The basic problem is that no amount of sampling can conclusively establish that a distribution has zero entropy.

THEOREM 6. *For every $\gamma > 1$, there is no algorithm that γ -approximates the entropy of every distribution in the generation oracle model.*

Proof. Let \mathcal{A} be any algorithm for computing entropy, and let $t(n)$ be an upper bound on its running time on distributions over $[n]$. Consider the two distributions $\mathbf{p} = \langle 1, 0, \dots, 0 \rangle$ and $\mathbf{q} = \langle 1 - 1/(100t(n)), 1/(100t(n)), 0, \dots, 0 \rangle$. Notice that \mathbf{p} has zero entropy while \mathbf{q} has positive entropy.

Suppose we run \mathcal{A} on either \mathbf{p} or \mathbf{q} . Since it uses at most $t(n)$ samples, its oracle calls will almost always (99% of the time) produce a succession of identical elements regardless of whether the underlying distribution is \mathbf{p} or \mathbf{q} . In such cases, if \mathcal{A} guesses that the entropy is zero, its approximation factor on \mathbf{q} will be unbounded, whereas if it guesses a positive number, its approximation factor on \mathbf{p} will be unbounded. \square

3.2.2. A lower bound on approximating the entropy of high-entropy distributions. The following theorem shows a lower bound on the number of samples required to approximate the entropy of distributions with high entropy.

THEOREM 7. *For every $\gamma > 1$ and sufficiently large n , any algorithm in the generation oracle model that γ -approximates the entropy of a distribution in $\mathcal{D}_{(\log n)/\gamma^2}$ requires $\Omega(n^{1/2\gamma^2})$ samples.*

Proof. Consider two distributions \mathbf{p} and \mathbf{q} on n elements, where \mathbf{p} is uniform on the set $[n]$ and \mathbf{q} is uniform on a randomly chosen subset $S \subseteq [n]$ of size n^{1/γ^2} . It is easy to see that $H(\mathbf{p})/H(\mathbf{q}) = \gamma^2$. By the birthday paradox, with probability $1/2$, we will not see any repetitions if we take $n^{1/2\gamma^2}$ samples from either distribution. In such cases, the samples from \mathbf{p} and \mathbf{q} appear identical. Thus at least $\Omega(n^{1/2\gamma^2})$ samples are needed to distinguish these distributions. \square

As we mentioned before, in the conference version of this paper, we claimed another lower bound of $\Omega(n^{2/(5\gamma^2-2)})$ for approximating the entropy, but retract this bound as there is a gap in our proof. Recently, however, Ron [13] showed a lower bound of $\tilde{\Omega}(n^{2/(6\gamma^2-3)})$ for approximating the entropy. This is better than the lower bound (in Theorem 7) of $\Omega(n^{1/2\gamma^2})$ when $\gamma < \sqrt{3}/2$. Her proof also exhibits two distributions with entropy ratio γ^2 and shows that the two distributions are indistinguishable unless $\tilde{\Omega}(n^{2/(6\gamma^2-3)})$ samples are taken.

4. The evaluation oracle model: A lower bound. In the introduction, we mentioned that, for general distributions over $[n]$, a linear number of queries is necessary to approximate the entropy in the evaluation oracle model. Since there are only n possible queries, the complexity of entropy approximation in this model is settled. Next, we study the number of queries needed when a lower bound on the entropy of the distribution can be assumed.

THEOREM 8. *Let $\gamma > 1, h > 0$, and n be sufficiently large. If an algorithm \mathcal{A} that operates in the evaluation oracle model achieves a γ -approximation to the entropy of distributions over $[n]$ in \mathcal{D}_h , then it must make $\Omega(n2^{-\gamma^2(h+1)})$ queries.*

Proof. We will define two distributions \mathbf{p} and \mathbf{q} in \mathcal{D}_h that have entropy ratio at least γ^2 and yet require $\Omega(n2^{-\gamma^2(h+1)})$ queries to distinguish.

Let R be a subset of $[n]$ of size $2^{\gamma^2(h+1)}$, chosen uniformly at random. Distribution \mathbf{p} is defined to be uniform over R . Let S also be a uniform-random subset of $[n]$, but of smaller size $\beta \cdot 2^{\gamma^2(h+1)}$, where $\beta = 1/(\gamma^2(h+1)/h)$. In addition, pick s randomly from $[n] \setminus S$. Distribution \mathbf{q} assigns probability $2^{-\gamma^2(h+1)}$ to each element in S and assigns the rest of the probability mass, namely, $1 - \beta$, to s .

Both these distributions belong to \mathcal{D}_h : $H(\mathbf{p}) = \gamma^2(h+1)$, and $H(\mathbf{q})$ is between h and $h+1$ (to see this, notice $H_S(\mathbf{q}) = h$). The ratio between their entropies is $H(\mathbf{p})/H(\mathbf{q}) \geq \gamma^2$.

In the evaluation oracle, any algorithm that distinguishes between \mathbf{p} and \mathbf{q} must (on at least one of these two inputs) discover some location $i \in [n]$ with nonzero probability. The number of queries required is therefore at least the reciprocal of the fraction of the elements with nonzero probabilities, which is $\Omega(n/2^{\gamma^2(h+1)})$. \square

5. The combined oracle model. In this section we consider the combined oracle model in which an algorithm is given both evaluation and generation oracle access to the same distribution.

5.1. Upper bound. The entropy of a distribution \mathbf{p} can be viewed as the expected value of $-\log p_i$, where i is distributed according to \mathbf{p} . This suggests an algorithm as follows:

1. Draw m samples from the generation oracle (m to be defined later). Call these i_1, \dots, i_m .

2. For each i_j , ask the evaluation oracle for p_{i_j} .
3. Return $-\frac{1}{m} \sum_{j=1}^m \log p_{i_j}$.

As we will now see, if $H(\mathbf{p})$ is not too small, this algorithm needs only a polylogarithmic number of queries in order to return a good approximation.

THEOREM 9. *Pick any $\gamma > 1$ and any $h > 0$. If the above algorithm is run with $m = O\left(\frac{\gamma^2 \log^2 n}{h^2(\gamma-1)^2}\right)$, then it returns a γ -approximation to the entropy of any distribution over $[n]$ in \mathcal{D}_h , with success probability at least $3/4$.*

Proof. Let $m \stackrel{\text{def}}{=} \frac{3\gamma^2 \log^2 n}{h^2(\gamma-1)^2}$. Define the random variable $X_j \stackrel{\text{def}}{=} -\log p_{i_j}$ for $j = 1, \dots, m$, and let $X = (1/m) \sum_j X_j$ be the final answer returned. Clearly, $E[X] = E[X_j] = H(\mathbf{p})$. All that needs to be shown is that the variance of X is not too large. Since the X_j 's are independent, it will suffice to bound the variance of an individual X_j .

LEMMA 10. $\text{Var}[X_j] \leq \log^2 n$.

Proof. For $n = 2$, maximizing $\text{Var}[X_j] = p \log^2 p + (1-p) \log^2(1-p) - (p \log p + (1-p) \log(1-p))^2$ subject to $0 \leq p \leq 1$ yields $\text{Var}[X_j] < 1 = \log^2 n$. Therefore, let $n \geq 3$. Since $\text{Var}[X_j] \leq E[X_j^2]$, it suffices to show an upper bound on $E[X_j^2] = \sum_i p_i \log^2 p_i$.

Note that the function $f(x) = x \log^2 x$ is concave for $0 < x < e^{-1}$. Hence, $\sum_i f(p_i)$ is a symmetric concave function when its domain is limited to $\mathbf{p} \in (0, 1/e)^n$ and, as in Lemma 1, is maximized (on this domain) when \mathbf{p} is uniform. This maximum value is $\log^2 n$.

To finish the proof, we need to show that we cannot attain higher values of $\sum_i f(p_i)$ by looking at $\mathbf{p} \notin (0, 1/e)^n$. To this end, suppose $p_j \geq e^{-1}$ for some j . Then there exists k such that $p_k \leq (1 - p_j)/(n - 1)$. Consider the derivative $f'(x) = \log^2 x + 2 \log(e) \log x$ at points p_j and p_k . Using simple calculus and the fact that $n \geq 3$, it is easy to check that $f'(p_k) > f'(p_j)$. Hence, we can increase the sum by simultaneously decreasing p_j and increasing p_k . By combining this result with the argument above, we conclude that $\sum_i f(p_i) \leq \log^2 n$. \square

Since the X_j 's are identical and independent, $\text{Var}[X] = \text{Var}[X_j]/m \leq (\log^2 n)/m$. To bound the error probability of our algorithm, we now use Chebyshev's inequality, which states that, for any $\rho > 0$,

$$\Pr[|X - E[X]| \geq \rho] \leq \text{Var}[X]/\rho^2.$$

Hence, we get

$$\begin{aligned} \Pr[\text{A does not } \gamma\text{-approximate } H(\mathbf{p})] &= \Pr\left[X \leq \frac{H(\mathbf{p})}{\gamma} \text{ or } X \geq \gamma \cdot H(\mathbf{p})\right] \\ &\leq \Pr\left[|X - H(\mathbf{p})| \geq \frac{(\gamma - 1) \cdot H(\mathbf{p})}{\gamma}\right] \\ &\leq \frac{\gamma^2 \log^2 n}{m \cdot H(\mathbf{p})^2 (\gamma - 1)^2} \leq \frac{1}{3}, \end{aligned}$$

where the last inequality follows from the choice of m . \square

COROLLARY 11. *There exists an algorithm \mathcal{A} in the combined oracle model that γ -approximates $H(\mathbf{p})$ in $O\left(\frac{\gamma}{\gamma-1}\right)^2$ time for distributions with $H(\mathbf{p}) = \Omega(\log n)$.*

5.2. Lower bounds. This next theorem gives a lower bound for the combined oracle model when the entropy of the distribution is allowed to be very small—so small that, for instance, the previous upper bound does not apply.

THEOREM 12. *Pick any $\gamma > 1$ and any sufficiently large n . Then any algorithm in the combined oracle model that γ -approximates the entropy of distributions over $[n]$ (with nonzero entropy) must make $\Omega(n^{1/\gamma^2})$ oracle calls.*

Proof. Let $\alpha = \frac{1}{\gamma^2} - \frac{\log e}{\log n} < 1$. Consider distributions \mathbf{p} and \mathbf{q} defined as follows:

$$p_i \stackrel{\text{def}}{=} \begin{cases} 1 - n^{-\alpha} & i = 1, \\ n^{-\alpha} & i = 2, \\ 0 & \text{otherwise,} \end{cases}$$

$$q_i \stackrel{\text{def}}{=} \begin{cases} 1 - n^{-\alpha} & i = 1, \\ n^{-1} & 2 \leq i \leq n^{1-\alpha} + 1, \\ 0 & \text{otherwise.} \end{cases}$$

Note that, by the concavity of $f(x) = -x \log x$ for $0 \leq \delta < 1$, and that $f'(1) = -\log e$, we have that $-(1 - \delta) \log(1 - \delta) \leq \delta \log e$. Hence, a quick calculation shows that $H(\mathbf{p}) = -(1 - n^{-\alpha}) \log(1 - n^{-\alpha}) + n^{-\alpha} \log n^\alpha \leq n^{-\alpha} (\log e + \alpha \log n)$ and $H(\mathbf{q}) > n^{-\alpha} \log n$. By the choice of α , $H(\mathbf{q})/H(\mathbf{p}) > \gamma^2$.

Let \mathcal{P} be the family of distributions obtained from \mathbf{p} by permuting the labels of the elements. Define \mathcal{Q} similarly for \mathbf{q} . It is simple to verify that any algorithm taking $o(n^\alpha)$ samples and making $o(n^\alpha)$ probes will fail to distinguish between a randomly chosen member of \mathcal{P} and a randomly chosen member of \mathcal{Q} with high probability. To finish, notice that $n^\alpha = e^{-1} n^{1/\gamma^2}$. \square

The next theorem gives a lower bound on the complexity of approximating the entropy in the combined oracle model for distributions with entropy above some specific threshold. The proof generalizes the counterexample in Theorem 12.

THEOREM 13. *Pick any $\gamma > 1$, any $h > 0$, and any sufficiently large n . Then any algorithm in the combined oracle model that γ -approximates the entropy of distributions over $[n]$ in \mathcal{D}_h must make $\Omega(\log n / (h(\gamma^2 - 1) + 2\gamma^2))$ oracle calls.*

Proof. Let $w = (h(\gamma^2 - 1) + 2\gamma^2) / \log n$ and $k \stackrel{\text{def}}{=} \lceil 2^{h/(1-w)} \rceil$. Consider the following distributions \mathbf{p} and \mathbf{q} :

$$p_i \stackrel{\text{def}}{=} \begin{cases} (1 - w)/k & 1 \leq i \leq k, \\ w & i = k + 1, \\ 0 & \text{otherwise,} \end{cases}$$

$$q_i \stackrel{\text{def}}{=} \begin{cases} (1 - w)/k & 1 \leq i \leq k, \\ n^{-1} & k + 1 \leq i \leq k + wn, \\ 0 & \text{otherwise.} \end{cases}$$

Note that $H(\mathbf{p}) = (1 - w) \log \frac{k}{1-w} - w \log w = (1 - w) \log k - (1 - w) \log(1 - w) - w \log w$. Hence, $h \leq H(\mathbf{p}) \leq h + 2$. Similarly, $H(\mathbf{q}) > h + w \log n$.

Let \mathcal{P} be the family of distributions obtained from \mathbf{p} by permuting the labels of the elements. Define \mathcal{Q} similarly for \mathbf{q} . It is simple to verify that any algorithm taking $o(1/w)$ samples and making $o(1/w)$ probes will fail to distinguish between a randomly chosen member of \mathcal{P} and a randomly chosen member of \mathcal{Q} with high probability.

Meanwhile, by the choice of w , the entropy ratio is

$$\frac{H(\mathbf{q})}{H(\mathbf{p})} > \frac{h + w \log n}{h + 2} = \frac{h\gamma^2 + 2\gamma^2}{h + 2} = \gamma^2.$$

This concludes the proof. \square

6. Monotone distributions. A *monotone distribution* $\mathbf{p} = \langle p_1, \dots, p_n \rangle$ is one for which $p_i \geq p_{i+1}$ for all i . The structure of a monotone distribution makes it much easier to approximate the entropy.

6.1. The evaluation oracle model. We show that given evaluation oracle access to a monotone distribution, we can approximate the entropy in polylogarithmic time.

THEOREM 14. *For any $\gamma > 1$, there is an algorithm in the evaluation oracle model that γ -approximates the entropy of monotone distributions on $[n]$ in the family $\mathcal{D}_{\Omega(\gamma^2/(\sqrt{\gamma}-1))}$ and runs in $O(\lceil 1/\log \gamma \rceil \log n)$ time.*

Proof. Algorithm \mathcal{A} partitions the domain $[n]$ into intervals and only queries the endpoints of each interval. The remaining probability values are interpolated from these queries.

The partition of $[n]$ is constructed recursively, starting with a single *active* interval $[1, n]$ as follows:

While there is some active interval $[\ell, u]$:

- Make it inactive.
- If $p_\ell > n^{-2}$ and $p_\ell/p_u > \gamma$, then split $[\ell, u]$ into two equal-sized active subintervals.

Any required probability values (i.e., p_ℓ, p_u at each iteration) are obtained from the oracle. At the end of the procedure, the algorithm has probabilities for a particular sequence of elements $1 = i_0 \leq i_1 \leq \dots \leq i_k = n$, such that for each $j < k$, either $p_{i_j} \leq n^{-2}$ or $p_{i_j}/p_{i_{j+1}} \leq \gamma$. The splitting criteria ensure that the total number of queries $k + 1$ is roughly logarithmic in the number of elements; more precisely, $k \leq 1 + (1 + 4/\log \gamma) \log n$.

The algorithm then approximates \mathbf{p} by a distribution \mathbf{q} that is interpolated from the handful of p_i values that it queries as follows:

- For each i_j , set $q_{i_j} = p_{i_j}$.
- For any $i \in (i_j, i_{j+1})$: if $p_{i_j} \leq n^{-2}$, then set $q_i = 0$; otherwise set $q_i = \sqrt{p_{i_j} p_{i_{j+1}}}$.

Let B_0 denote the elements whose probabilities get set to zero, and let $B = [n] \setminus B_0$ be the remaining elements. We know that for $i \in B_0$, $p_i \leq n^{-2}$. Thus, $w_{\mathbf{p}}(B_0) \leq n^{-1}$ and, by Lemma 1(b), B_0 doesn't contribute much to the entropy: $H_{B_0}(\mathbf{p}) \leq 2n^{-1} \log n$. We therefore need to focus on B .

For each $i \in B$, define $c_i \stackrel{\text{def}}{=} q_i/p_i$. Since the endpoints of the interval containing i have probabilities that are within a multiplicative factor γ of each other, it follows that $\frac{1}{\sqrt{\gamma}} \leq c_i \leq \sqrt{\gamma}$. This means that $H_B(\mathbf{q})$ is not too different from $H_B(\mathbf{p})$:

$$\begin{aligned} H_B(\mathbf{q}) &= - \sum_{i \in B} q_i \log q_i = - \sum_{i \in B} c_i p_i \log(c_i p_i) = - \sum_{i \in B} c_i p_i \log p_i - \sum_{i \in B} c_i p_i \log c_i \\ &\leq \sqrt{\gamma} \cdot H_B(\mathbf{p}) + \frac{w_{\mathbf{p}}(B) \log e}{e} \leq \gamma \cdot H(\mathbf{p}) \end{aligned}$$

when $H(p) \geq \log e/(e(\gamma - \sqrt{\gamma}))$. The first inequality follows from the fact (see Lemma 1) that $-x \log x \leq (\log e)/e$ for all $x \in (0, 1)$. Similarly,

$$\begin{aligned} H_B(\mathbf{q}) &= - \sum_{i \in B} q_i \log q_i = - \sum_{i \in B} c_i p_i \log(c_i p_i) = - \sum_{i \in B} c_i p_i \log p_i - \sum_{i \in B} c_i p_i \log c_i \\ &\geq \frac{1}{\sqrt{\gamma}} \cdot H_B(\mathbf{p}) - w_{\mathbf{p}}(B) \sqrt{\gamma} \log \sqrt{\gamma} \geq \frac{1}{\sqrt{\gamma}} \left(H(\mathbf{p}) - \frac{2 \log n}{n} \right) - w_{\mathbf{p}}(B) \sqrt{\gamma} \log \sqrt{\gamma} \\ &\geq H(\mathbf{p})/\gamma \end{aligned}$$

when $H(\mathbf{p}) \geq (\gamma^2 + (2n^{-1}\sqrt{\gamma} \log n))/(\sqrt{\gamma} - 1)$. The second-to-last inequality uses $H_{B_0}(\mathbf{p}) \leq 2n^{-1} \log n$.

The algorithm outputs $H(\mathbf{q}) = H_B(\mathbf{q})$, which we've shown is a γ -approximation to $H(\mathbf{p})$. \square

6.2. The generation oracle model. We show that the entropy of a monotone distribution can also be approximated in polylogarithmic time in the generation oracle model. Our algorithm rests upon the following observation that is formally stated in Lemma 15: if a monotone distribution \mathbf{p} over $[n]$ is such that $w_{\mathbf{p}}([n/2])$ and $w_{\mathbf{p}}([n] \setminus [n/2])$ are very close, then the distribution must be close to uniform. In such a case, we can approximate the entropy of the distribution by the entropy of the uniform distribution.

The main idea behind our algorithm is to recursively partition the domain into halves, stopping the recursion when either (1) the probability masses of two halves are very close or (2) they are both too small to contribute much to the total entropy. Our algorithm can be viewed as forming a tree based on the set of samples S , where the root is labeled by the range $[1, n]$, and if the node labeled by the range $[i, j]$ is partitioned, its children are labeled by the ranges $[i, (i+j)/2]$ and $[(i+j)/2 + 1, j]$, respectively. Once the partition tree is determined, the algorithm estimates the entropy by summing the contributions from each leaf, assuming that the conditional distribution within a leaf (that is, the distribution restricted to the leaf's range) is uniform. By the choice of our splitting and stopping criteria, we show that the number of leaves in the tree is at most polylogarithmic in n . This in turn allows us to bound both the running time and the probability of error.

More specifically, the procedure **BuildTree** (S, β) takes as input a parameter $\beta > 1$ and a multiset S of m samples from \mathbf{p} and outputs a rooted binary tree T_S as follows: Let v be a node in the tree that is currently a leaf corresponding to the interval $[i, j]$ for some $i < j$. For an interval I , let S_I denote the set of samples that fall in I , and let $|I|$ denote the length of the interval. We determine that v will remain a leaf if either of the following two conditions is satisfied:

- $|S_{[i,j]}| < m\beta/\log^3 n$ (call v *light*), or
- $|S_{[i, \lfloor (i+j)/2 \rfloor]}| \leq \beta |S_{[\lfloor (i+j)/2 \rfloor + 1, j]}|$ (call v *balanced*).

Otherwise, we split v 's interval by attaching two children to v , corresponding to the intervals $[i, \lfloor (i+j)/2 \rfloor]$ (the left child) and $[\lfloor (i+j)/2 \rfloor + 1, j]$ (the right child). Let $\mathcal{I}(T_S)$ denote the set of intervals corresponding to the balanced leaves of T_S .

For each balanced interval $I \in \mathcal{I}(T_S)$, we estimate the contribution of the interval to the total entropy of the distribution. Note that if the interval I had uniform conditional distribution, then

$$\begin{aligned} H_I(\mathbf{p}) &= \sum_{i \in I} \frac{w_{\mathbf{p}}(I)}{|I|} \log \frac{|I|}{w_{\mathbf{p}}(I)} \\ &= w_{\mathbf{p}}(I) (\log |I| - \log w_{\mathbf{p}}(I)) \\ &= w_{\mathbf{p}}(I) \left(\log \left(\frac{|I|}{2} \right) - \log \left(\frac{w_{\mathbf{p}}(I)}{2} \right) \right). \end{aligned}$$

Motivated by this, we define a function $\alpha(I, \beta)$ that approximates the entropy in the balanced interval I :

$$\alpha(I, \beta) \stackrel{\text{def}}{=} \frac{|S_I|}{m} \left(\log \frac{|I|}{2} + \log \frac{m}{\beta |S_I|} \right).$$

We now give the top level description of our algorithm.

ALGORITHM MONOTONEAPPROXIMATEENTROPY (γ).

1. $\beta = \sqrt{\gamma}$.
2. Get a multiset S of $m = O((\beta^5 \log^4 n)/(\beta - 1)^2)$ samples from \mathbf{p} .
3. $T_S = \mathbf{BuildTree}(S, \beta)$.
4. Output $\sum_{I \in \mathcal{I}(T_S)} \alpha(I, \beta)$.

Overview of the proof. The main steps in the proof are the following. First, we give a key lemma on which the whole algorithm is based; this lemma implies that for an interval corresponding to a balanced leaf, the upper and lower bounds on the possible entropy values are fairly close (Lemma 15). The rest of the proof is devoted to showing that the domain can be split into intervals that are either balanced or small enough that they do not contribute much (in total) to the entropy of the distribution. In Lemma 16, we show that sampling can be reliably used to decide whether or not to split an interval. We then quantify the relationship between $\alpha(I, \beta)$ and $H_I(\mathbf{p})$ for each interval I corresponding to a balanced leaf, taking the sampling error into account (Lemma 18). Note that if it were possible to partition the whole domain into balanced intervals of large enough size, then it would be a simple matter to bound the number of intervals and thus the error probability and running time of the algorithm. The most challenging part of the proof is to deal with the light intervals, in particular to show the following two properties: (1) the number of such intervals is approximately logarithmic in the size of the domain (Lemma 19), and (2) their total entropy contribution is negligible and thus can be ignored. To show these properties, we prove an interesting and nontrivial property of the tree T_S : at any level, it contains at most $O(\log \log n)$ nodes. Thereafter, (1) and (2) follow easily.

First, we show upper and lower bounds on the entropy contribution of an interval in terms of the total weight and the weight distribution between two halves of the interval.

LEMMA 15. *Let I be an interval of length $2k$ in $[n]$, let I_1 and I_2 be the bisection of I , and let \mathbf{p} be a monotone distribution over $[n]$. Then,*

$$H_I(\mathbf{p}) \leq w_{\mathbf{p}}(I) \log k - w_{\mathbf{p}}(I_1) \log w_{\mathbf{p}}(I_1) - w_{\mathbf{p}}(I_2) \log w_{\mathbf{p}}(I_2)$$

and

$$H_I(\mathbf{p}) \geq 2w_{\mathbf{p}}(I_2) \log k - w_{\mathbf{p}}(I_2) \left(\log w_{\mathbf{p}}(I_1) + \log w_{\mathbf{p}}(I_2) \right).$$

Notice in particular that the ratio of the upper bound to the lower bound is at most $w_{\mathbf{p}}(I)/2w_{\mathbf{p}}(I_2)$.

Proof. The upper bound follows from Lemma 1(a): the partial entropies $H_{I_1}(\mathbf{p})$ and $H_{I_2}(\mathbf{p})$ are maximized when their weights are spread uniformly over their constituent elements.

Let $w_1 \stackrel{\text{def}}{=} w_{\mathbf{p}}(I_1)$ and $w_2 \stackrel{\text{def}}{=} w_{\mathbf{p}}(I_2)$. We will prove the lower bound even for functions that satisfy a relaxation of the monotonicity property: namely, the condition that for $i \leq k$, $p_i \geq w_2/k$, and for $i > k$, $p_i \leq w_1/k$. It is easy to verify that any monotone distribution will satisfy this new constraint. A lower bound on $H_{I_1}(\mathbf{p})$ is given by Lemma 1(c) (plug in w_2/k for as many elements as possible), and for $H_{I_2}(\mathbf{p})$ it follows immediately from $p_i \leq w_1/k$ for $i \in I_2$. Combining, we get

$$H_I(\mathbf{p}) \geq w_2 \log \frac{k}{w_2} + w_2 \log \frac{k}{w_1}. \quad \square$$

For a balanced leaf corresponding to an interval I with bisection I_1, I_2 , the error in the entropy estimate depends upon the ratio $w_{\mathbf{p}}(I)/2w_{\mathbf{p}}(I_2)$. This can be made small by choosing the parameter β appropriately.

The following lemma shows that the samples can be used to decide if an interval should be split.

LEMMA 16. *Let I be an interval in $[1, n]$ such that $w_{\mathbf{p}}(I) \geq \log^{-3} n$, and let I_1, I_2 be a bisection of I . Let S be a sample set of size $m = O((\beta^5 \log^4 n)/(\beta - 1)^2)$ drawn from \mathbf{p} . For $\beta > 1$,*

1. *with probability at least $1 - n^{-2}$, $(1/\beta) \cdot m \cdot w_{\mathbf{p}}(I) \leq |S_I| \leq \beta \cdot m \cdot w_{\mathbf{p}}(I)$;*
2. *if $w_{\mathbf{p}}(I_1)/w_{\mathbf{p}}(I_2) \geq 2\beta - 1$, then with probability at least $1 - 2n^{-2}$, $|S_{I_1}| \geq \beta \cdot |S_{I_2}|$;*
3. *if $w_{\mathbf{p}}(I_1)/w_{\mathbf{p}}(I_2) \leq (1 + \beta)/2$, then with probability at least $1 - 2n^{-2}$, $|S_{I_1}| \leq \beta \cdot |S_{I_2}|$.*

Proof. Part 1 follows from a straightforward application of multiplicative Chernoff bounds. The random variable $|S_I|$ is the sum of m independent Bernoulli trials, each with success probability $w_{\mathbf{p}}(I)$. Therefore $\mathbb{E}[|S_I|] = mw_{\mathbf{p}}(I)$, and by the choice of m in the algorithm, the probability that $|S_I|$ deviates from its expectation by more than a multiplicative factor of β is at most $1/n^2$.

From part 1, we know that with probability at least $1 - n^{-2}$, $|S_I| \geq mw_{\mathbf{p}}(I)/\beta$. Fix any $t \geq mw_{\mathbf{p}}(I)/\beta$. To prove part 2, consider the ratio of the number of samples from I_1 and I_2 conditioned on the event that there are exactly t samples from I . Let Y_i , for $i = 1, \dots, t$, be an indicator random variable that takes the value 1 if the i th of these t samples is in I_2 , and $Y = \sum_i Y_i$. Therefore, we want to show that the probability that $(t - Y)/Y < \beta$ is at most $2/n^2$.

The rest of the proof is an application of Chernoff bounds. Note that $(t - Y)/Y < \beta$ implies $Y > t/(\beta + 1)$. Since $\mathbb{E}[Y] \leq t/(2\beta)$, we get

$$\Pr \left[Y > \frac{t}{\beta + 1} \right] \leq \Pr \left[Y > \mathbb{E}[Y] + \frac{t(\beta - 1)}{2\beta(\beta + 1)} \right] \leq \exp \left(\frac{-t(\beta - 1)^2}{\beta^2(\beta + 1)^2} \right).$$

Conditioned on the event that $t \geq mw_{\mathbf{p}}(I)/\beta$, this probability is less than $1/n^2$. Combining this with part 1, we can conclude that, with probability at least $1 - 2n^{-2}$, we have $|S_{I_1}| \geq \beta \cdot |S_{I_2}|$.

Similarly, the third part of the lemma can be proved. \square

There are various events that we would like to count on, for instance, that for balanced intervals I the ratio of the weights of the two halves is at most $2\beta - 1$, and that intervals associated with two sibling nodes have weight ratio at least $(1 + \beta)/2$. Lemma 16 tells us that these events hold with high probability. We now package all of them into a single assumption.

ASSUMPTION 17. (1) *For each interval I corresponding to a balanced node of the tree, $|S_I|$ lies in the range $[(1/\beta) \cdot m \cdot w_{\mathbf{p}}(I), \beta \cdot m \cdot w_{\mathbf{p}}(I)]$; (2) for each interval I we decide to split, $w_{\mathbf{p}}(I_1)/w_{\mathbf{p}}(I_2) \geq (1 + \beta)/2$; (3) for each balanced interval I , we have $w_{\mathbf{p}}(I_1)/w_{\mathbf{p}}(I_2) \leq 2\beta - 1$; and (4) each light leaf has weight at most $\beta^2/\log^3 n$.*

Now we can show that under the assumption above, the entropy contribution of each balanced interval is approximated well. Recall that $\mathcal{I}(T_S)$ is the set of all balanced intervals in T_S .

LEMMA 18. *Under Assumption 17, for every $I \in \mathcal{I}(T_S)$, if $w_{\mathbf{p}}(I) \geq \log^{-3} n$, then*

$$\frac{H_I(\mathbf{p})}{\beta} - 2\beta \cdot w_{\mathbf{p}}(I) \leq \alpha(I, \beta) \leq \beta^2 \cdot H_I(\mathbf{p}).$$

Proof. Let I_1, I_2 be the bisection of I . Under Assumption 17, $|S_I|/(m\beta) \leq w_{\mathbf{p}}(I) \leq |S_I|/\beta/m$ and $w_{\mathbf{p}}(I_1)/w_{\mathbf{p}}(I_2) \leq 2\beta - 1$. These imply that the upper and

lower bounds for $H_I(\mathbf{p})$ given in Lemma 15 are within a multiplicative factor β of each other. Therefore our entropy estimate $\alpha(I, \beta)$ is not too far from $H_I(\mathbf{p})$:

$$\begin{aligned} \alpha(I, \beta) &= \frac{|S_I|}{m} \left(\log \frac{|I|}{2} + \log \frac{m}{\beta |S_I|} \right) \\ &\leq \beta w_{\mathbf{p}}(I) \log \left(\frac{|I|}{2} \right) - \beta w_{\mathbf{p}}(I) \log w_{\mathbf{p}}(I) \\ &\leq \beta \cdot \left(w_{\mathbf{p}}(I) \log \left(\frac{|I|}{2} \right) - w_{\mathbf{p}}(I_1) \log w_{\mathbf{p}}(I_1) - w_{\mathbf{p}}(I_2) \log w_{\mathbf{p}}(I_2) \right) \\ &\leq \beta^2 \cdot H_I(\mathbf{p}). \end{aligned}$$

The second inequality above is a simple consequence of $w_{\mathbf{p}}(I) = w_{\mathbf{p}}(I_1) + w_{\mathbf{p}}(I_2)$, and the expression on that line is exactly (β times) the upper bound of Lemma 15. Similarly, for the other direction,

$$\begin{aligned} \alpha(I, \beta) &= \frac{|S_I|}{m} \left(\log \frac{|I|}{2} + \log \frac{m}{\beta |S_I|} \right) \\ &\geq \frac{w_{\mathbf{p}}(I)}{\beta} \log \frac{|I|}{2} - \frac{w_{\mathbf{p}}(I)}{\beta} \log \frac{w_{\mathbf{p}}(I)}{2} - \frac{w_{\mathbf{p}}(I)}{\beta} \log 2\beta^2 \\ &\geq \frac{1}{\beta} \left(w_{\mathbf{p}}(I) \log \frac{|I|}{2} - w_{\mathbf{p}}(I_1) \log w_{\mathbf{p}}(I_1) - w_{\mathbf{p}}(I_2) \log w_{\mathbf{p}}(I_2) \right) - 2\beta \cdot w_{\mathbf{p}}(I) \\ &\geq \frac{H_I(\mathbf{p})}{\beta} - 2\beta \cdot w_{\mathbf{p}}(I). \end{aligned}$$

The second inequality follows from the concavity of $\log x$. \square

Next, we show a bound on the number of nodes in the tree.

LEMMA 19. *Under Assumption 17, given $\beta > 1$, the number of nodes in T_S is at most*

$$\frac{12 \log n \log \log n}{\log(\beta + 1) - 1}.$$

Proof. For any given level of the tree, let v_1, \dots, v_{2k} denote the internal (that is, nonleaf) nodes at that level, ordered by the intervals they define. There is an even number of these nodes because each has a sibling at the same level. If v_i, v_{i+1} are siblings, we know from Assumption 17 that $w(v_i) \geq w(v_{i+1}) \cdot (1 + \beta)/2$. In general, by monotonicity, $w(v_i) \geq w(v_{i+1})$. Therefore, as one moves from v_1 to v_{2k} , the weight $w(v_i)$ drops by a factor of at least $(1 + \beta)/2$ for every two nodes. Moreover, these weights never drop below $1/\log^3 n$, by the split criterion and Assumption 17. It follows that

$$k \leq \frac{3 \log \log n}{\log(1 + \beta) - 1}.$$

We now have a bound on the number of internal nodes at any level. To finish the lemma, we observe that there are at most $\log n$ levels, that the total number of nodes (internal and leaf) is twice the number of internal nodes plus one, and that we have overcounted by at least one at the root level. \square

Now, we are ready to complete our proof.

THEOREM 20. *For every $\gamma > 1$, there is an algorithm that approximates the entropy of a monotone distribution on $[n]$ in $\mathcal{D}_{(6\gamma^{3/2}/(\log(\sqrt{\gamma}+1)-1)(\sqrt{\gamma}-1))}$ to within a multiplicative factor of γ with probability at least $3/4$ in*

$$O\left(\frac{\gamma^{5/2} \log^6 n}{(\sqrt{\gamma}-1)^2(\log(\sqrt{\gamma}+1)-1)}\right) \text{ time.}$$

Proof. Suppose Assumption 17 holds; we will come back and address this later. Let's start by handling the leaves. By Assumption 17, each light leaf has weight at most $\beta^2/\log^3 n$, and so by Lemma 19, the total weight of the intervals associated with light leaves is at most

$$\frac{6\beta^2 \log \log n}{(\log(\beta+1)-1) \log^2 n}.$$

Therefore, their combined entropy contribution is at most $\log n$ times this,

$$\frac{6\gamma \log \log n}{(\log(\sqrt{\gamma}+1)-1) \log n}$$

(recall $\beta^2 = \gamma$), which will turn out to be negligible for our purposes.

Now we move on to the internal nodes. By Lemma 18,

$$\frac{H_I(\mathbf{p})}{\beta} - 2\beta \cdot w_{\mathbf{p}}(I) \leq \alpha(I, \beta) \leq \beta^2 \cdot H_I(\mathbf{p})$$

for each interval I associated with a balanced leaf. Let $B = \cup_{I \in \mathcal{I}(T_S)} I$. The algorithm's output is

$$\sum_{I \in \mathcal{I}(T_S)} \alpha(I, \beta) \leq \sum_{I \in \mathcal{I}(T_S)} \beta^2 \cdot H_I(\mathbf{p}) = \gamma \cdot H_B(\mathbf{p}) \leq \gamma \cdot H(\mathbf{p}).$$

We can show the other direction as follows:

$$\sum_{I \in \mathcal{I}(T_S)} \alpha(I, \beta) \geq \frac{H_B(\mathbf{p})}{\beta} - 2\beta \geq \frac{H(\mathbf{p}) - \frac{6\gamma \log \log n}{(\log(\sqrt{\gamma}+1)-1) \log n}}{\beta} - 2\beta \geq \frac{H(\mathbf{p})}{\beta^2}$$

when $H(\mathbf{p}) \geq (6\gamma^{3/2}/(\log(\sqrt{\gamma}+1)-1)(\sqrt{\gamma}-1))$.

We now proceed to justify Assumption 17. Consider the $2n$ intervals that correspond to the nodes of a complete tree T . By Lemma 16, Assumption 17 fails to hold for a particular interval of T with probability $O(1/n^2)$. Hence, Assumption 17 fails to hold for T_S with probability $O(1/n)$ by the union bound over all the intervals. Therefore, the error probability of the algorithm is $o(1)$. The running time of the algorithm is the sample size times the size of T_S . \square

Note that the lower bound shown in Theorem 6 applies to monotone distributions. Therefore, a restriction on the entropy such as the one in the statement of Theorem 20 is necessary.

7. Subset-uniform distributions. Consider the family of distributions \mathcal{E}_k that are uniform over some subset $K \subseteq [n]$ with $|K| = k$. The entropy of this class of distributions is $\log k$. If we approximate k to within a multiplicative factor of γ , then we get a very strong additive approximation to $\log k$. Now, given a generation

oracle access to a distribution that is promised to be from \mathcal{E}_k for some k , the entropy estimation problem reduces to approximating k .

THEOREM 21. *For every $\gamma > 1$, there exists an algorithm in the generation oracle model that, for every k and for any distribution $\mathbf{p} \in \mathcal{E}_k$, outputs ℓ such that $k/\gamma \leq \ell \leq \gamma k$ with probability at least $3/4$ in $O(\gamma\sqrt{k}/(\gamma - 1))$ time.*

Proof. Our algorithm, inspired by [5], is as follows.

1. Let $c = 16\gamma/(\gamma - 1)^2$.
2. Draw samples until at least c pairwise collisions are observed.
3. If M is the number of samples seen, output $\binom{M}{2}/c$.

Note that M is a random variable.

To analyze this algorithm, pick any integer m and suppose that m samples are drawn from the distribution. For $i < j$, let X_{ij} be an indicator random variable denoting a collision between the i th and j th samples seen. Let $S_m = \sum_{i < j} X_{ij}$ be the total number of collisions.

For any $i < j$, $E[X_{ij}] = 1/k$; therefore $E[S_m] = \binom{m}{2} \cdot 1/k$. This motivates the algorithm above. To bound the chance of failure, we also need the variance of S_m . Notice that

$$E[S_m^2] = E\left[\left(\sum_{i < j} X_{ij}\right)\left(\sum_{a < b} X_{ab}\right)\right] = \sum_{i < j, a < b} E[X_{ij}X_{ab}].$$

In the final summation, the various terms can be segregated according to the cardinality of the set $\{i, j, a, b\}$. If this set has cardinality 3 or 4, then $E[X_{ij}X_{ab}] = 1/k^2$. If the set has cardinality 2, then $E[X_{ij}X_{ab}] = 1/k$. This last possibility occurs for exactly $\binom{m}{2}$ of the $\binom{m}{2}^2$ terms in the summation. Therefore

$$E[S_m^2] = \left(\binom{m}{2}^2 - \binom{m}{2}\right) \frac{1}{k^2} + \binom{m}{2} \frac{1}{k},$$

whereupon $\text{Var}[S_m] = E[S_m^2] - E[S_m]^2 = \binom{m}{2}(1/k - 1/k^2) \leq E[S_m]$.

What is the chance that the algorithm outputs a number less than k/γ ? Let m_0 be the largest integer m such that $\binom{m}{2} < ck/\gamma$:

$$\begin{aligned} \Pr[\text{Output is } < k/\gamma] &= \Pr[M \leq m_0] = \Pr[S_{m_0} \geq c] \\ &\leq \Pr[|S_{m_0} - E[S_{m_0}]| \geq (c - E[S_{m_0}])]. \end{aligned}$$

This last probability can be bounded by Chebyshev's inequality, giving

$$\Pr[\text{Output is } < k/\gamma] \leq \frac{\text{Var}[S_{m_0}]}{(c - E[S_{m_0}])^2} \leq \frac{E[S_{m_0}]}{(c - E[S_{m_0}])^2} \leq \frac{\gamma}{c(\gamma - 1)^2} \leq \frac{1}{16},$$

where the last two inequalities follow from $E[S_{m_0}] < c/\gamma$ and from the particular choice of c .

To bound that chance that the output is more than $k\gamma$, we proceed similarly, letting m_0 denote the smallest integer m for which $\binom{m+1}{2} > c\gamma k$. Then,

$$\begin{aligned} \Pr[\text{Output is } > k\gamma] &= \Pr[M > m_0] = \Pr[S_{m_0} < c] \\ &\leq \Pr[|S_{m_0} - E[S_{m_0}]| \geq (E[S_{m_0}] - c)]. \end{aligned}$$

Again using Chebyshev's inequality, we get

$$\Pr[\text{Output is } > k\gamma] \leq \frac{\text{Var}[S_{m_0}]}{(\mathbb{E}[S_{m_0}] - c)^2} \leq \frac{\mathbb{E}[S_{m_0}]}{(\mathbb{E}[S_{m_0}] - c)^2} \leq \frac{3\gamma}{c(\gamma - 1)^2} \leq \frac{3}{16}.$$

The total probability of error is therefore at most $1/4$. When the algorithm succeeds, $\binom{M}{2}/c \leq k\gamma$, and so the number of samples (and the running time) is $O(\sqrt{ck\gamma})$. \square

Acknowledgment. We thank the referee for improving the presentation of the paper.

REFERENCES

- [1] A. ANTOS AND I. KONTOYIANNIS, *Estimating the entropy of discrete distributions*, in Proceedings of the IEEE International Symposium on Information Theory (ISIT), IEEE, Los Alamitos, CA, 2001, p. 45.
- [2] T. BATU, E. FISCHER, L. FORTNOW, R. KUMAR, R. RUBINFELD, AND P. WHITE, *Testing random variables for independence and identity*, in Proceedings of the 42nd Annual Symposium on Foundations of Computer Science, IEEE, Los Alamitos, CA, 2001, pp. 442–451.
- [3] T. BATU, L. FORTNOW, R. RUBINFELD, W. D. SMITH, AND P. WHITE, *Testing that distributions are close*, in Proceedings of the 41st Annual Symposium on Foundations of Computer Science, Redondo Beach, CA, IEEE, Los Alamitos, CA, 2000, pp. 259–269.
- [4] H. CAI, S. R. KULKARNI, AND S. VERDÚ, *Universal entropy estimation via block sorting*, IEEE Trans. Inform. Theory, 50 (2004), pp. 1551–1561.
- [5] O. GOLDREICH AND D. RON, *On Testing Expansion in Bounded-Degree Graphs*, Tech. report TR00-020, Electronic Colloquium on Computational Complexity (ECCC), <http://eccc.univ-trier.de/eccc-reports/2000/TR00-020/index.html> (2000).
- [6] O. GOLDREICH AND S. VADHAN, *Comparing entropies in statistical zero knowledge with applications to the structure of SZK*, in Proceedings of the 14th Annual IEEE Conference on Computational Complexity (CCC-99), May 4–6, 1999, IEEE, Los Alamitos, CA, pp. 54–73.
- [7] P. GRASSBERGER, *Entropy estimates from insufficient samplings*, E-print Physics/0307138, http://arxiv.org/PS_cache/physics/pdf/0307/0307138.pdf (2003).
- [8] B. HARRIS, *The statistical estimation of entropy in the non-parametric case*, in Topics in Information Theory, Colloq. Math. Soc., János Bolyai, 16, North-Holland, Amsterdam, 1977, pp. 323–355.
- [9] M. KEARNS, Y. MANSOUR, D. RON, R. RUBINFELD, R. E. SCHAPIRE, AND L. SELLIE, *On the learnability of discrete distributions*, in Proceedings of the 26th Annual Symposium on the Theory of Computing, ACM, New York, 1994, pp. 273–282.
- [10] S-K. MA, *Calculation of entropy from data of motion*, J. Statist. Phys., 26 (1981), pp. 221–240.
- [11] L. PANINSKI, *Estimation of entropy and mutual information*, Neural Computation, 15 (2003), pp. 1191–1253.
- [12] L. PANINSKI, *Estimating entropy on m bins given fewer than m samples*, IEEE Trans. Inform. Theory, 50 (2004), pp. 2200–2203.
- [13] D. RON, *Private communication*, 2004.
- [14] S. P. STRONG, R. KOBERLE, R. R. DE RUYTER VAN STEVENINCK, AND W. BIALEK, *Entropy and information in neural spike trains*, Phys. Rev. Lett., 80 (1998), pp. 197–200.
- [15] D. WOLPERT AND D. R. WOLF, *Estimating functions of probability distributions from a finite set of samples. Part I. Bayes estimators and the Shannon entropy*, Phys. Rev. E, 52 (1995), pp. 6841–6854.
- [16] A. J. WYNER AND D. FOSTER, *On the lower limits of entropy estimation*, IEEE Trans. Inform. Theory, submitted.

WELL-SEPARATED PAIR DECOMPOSITION FOR THE UNIT-DISK GRAPH METRIC AND ITS APPLICATIONS*

JIE GAO[†] AND LI ZHANG[‡]

Abstract. We extend the classic notion of well-separated pair decomposition [P. B. Callahan and S. R. Kosaraju, *J. ACM*, 42 (1975), pp. 67–90] to the unit-disk graph metric: the shortest path distance metric induced by the intersection graph of unit disks. We show that for the unit-disk graph metric of n points in the plane and for any constant $c \geq 1$, there exists a c -well-separated pair decomposition with $O(n \log n)$ pairs, and the decomposition can be computed in $O(n \log n)$ time. We also show that for the unit-ball graph metric in k dimensions where $k \geq 3$, there exists a c -well-separated pair decomposition with $O(n^{2-2/k})$ pairs, and the bound is tight in the worst case. We present the application of the well-separated pair decomposition in obtaining efficient algorithms for approximating the diameter, closest pair, nearest neighbor, center, median, and stretch factor, all under the unit-disk graph metric.

Key words. well-separated pair decomposition, unit-disk graph, approximation algorithm

AMS subject classifications. 68W25, 68W40, 68R10, 05C85

DOI. 10.1137/S0097539703436357

1. Introduction. Well-separated pair decomposition, introduced by Callahan and Kosaraju [10], has found numerous applications in solving proximity problems for points in the Euclidean space [8, 10, 9, 5, 4, 29, 25, 19, 14]. A pair of point sets (A, B) is *c-well-separated* if the distance between A, B is at least c times the diameters of both A and B . A well-separated pair decomposition of a point set consists of a set of well-separated pairs that cover all the pairs of distinct points, i.e., any two distinct points belong to the different sets of some pair. In [10], Callahan and Kosaraju showed that for any point set in an Euclidean space and for any constant $c \geq 1$, there always exists a c -well-separated pair decomposition with linearly many pairs. This fact has been very useful in obtaining nearly linear time algorithms for many problems, such as computing k -nearest neighbors, N -body potential fields, geometric spanners, and approximate minimum spanning trees. Well-separated pair decomposition is also shown to be very useful in obtaining efficient dynamic, parallel, and external memory algorithms [8, 9, 10, 7, 18].

The definition of well-separated pair decomposition can be naturally extended to any metric space. Curiously enough, however, there has been no work for such an extension. A possible reason is that a general metric space may not admit a well-separated pair decomposition with a subquadratic size. Indeed, even for the metric induced by a star tree with unit weight on each edge,¹ any well-separated pair decomposition requires quadratically many pairs. This makes the well-separated pair decomposition useless for such a metric. In this paper, we will show that for a certain metric, there do exist well-separated pair decompositions with almost linear size, and

*Received by the editors October 16, 2003; accepted for publication (in revised form) February 2, 2005; published electronically October 3, 2005.

<http://www.siam.org/journals/sicomp/35-1/43635.html>

[†]Center for the Mathematics of Information, California Institute of Technology, Pasadena, CA 91125 (jgao@ist.caltech.edu). This work was done while the author was at the Department of Computer Science, Stanford University, Stanford, CA 94305.

[‡]Hewlett-Packard Labs, 1501 Page Mill Road, Palo Alto, CA 94304 (l.zhang@hp.com).

¹A metric induced by a graph (with positive edge weights) is the shortest path distance metric of the graph.

therefore many proximity problems under that metric can be solved efficiently. The metric we investigate is the so-called unit-disk graph metric. At the same time we call the well-separated pair decomposition in the Euclidean space the *geometric well-separated pair decomposition*, to be distinguished from the decomposition in graph metrics.

For a point set S in the plane, its unit-disk graph [12] is formed by connecting two points p, q in S if the Euclidean distance $d(p, q)$ is at most 1. A unit-disk graph can also be viewed as the intersection graph of a set of unit disks centered at the points in S . We consider the weighted unit-disk graphs where each edge (p, q) receives the weight $d(p, q)$. Similarly, unit-ball graphs can be defined for points in high dimensions. Such graphs have been used extensively to model the communication or influence between objects [27, 21] and studied in many different contexts [12, 6, 22, 15]. For example, wireless ad hoc networks can be modeled by unit-disk graphs [23, 30, 31], as two wireless nodes can directly communicate with each other only if they are within a certain distance. In unsupervised learning, for a dense sampling of points from some unknown manifold, the length of the shortest path on the unit-ball graph is a good approximation of the geodesic distance on the underlying (unknown) manifold if the radius is chosen appropriately [34, 16]. In this paper, we show that the all-pairs shortest path lengths of nodes in unit-disk graphs (or unit-ball graphs) can be compactly encoded and efficiently estimated by a subquadratic size well-separated pair decomposition.

2. Overview. In this paper, we show that for the metric induced by the unit-disk graph on n points and for any constant $c \geq 1$, there does exist a c -well-separated pair decomposition with $O(n \log n)$ pairs, and such a decomposition can be computed in $O(n \log n)$ time. We also show that the bounds can be extended to higher dimensions: for $k \geq 3$, there always exists a c -well-separated pair decomposition with size $O(n^{2-2/k})$ for the unit-ball graph metric on n points, and the bound is tight in the worst case. The construction time is $O(n^{4/3} \text{polylog } n)$ for $k = 3$ and $O(n^{2-2/k})$ for $k \geq 4$.

The difficulty in obtaining a well-separated pair decomposition for unit-disk graph metric is that two points that are close in the space are not necessarily close under the graph metric. We first prove the bound for the point set with constant-bounded density, i.e., a point set where any unit disk covers only a constant number of points in the set, by using a packing argument similar to the one in [20]. For a point set with unbounded density, we apply the clustering technique similar to the one used in [17] to the point set and obtain a set of clusterheads with a bounded density. We then apply the result for bounded density point set on those clusterheads. Then, by combining the well-separated pair decomposition for the bounded density point sets and for the Euclidean metric, we are able to show that the bound holds for any point sets.

For a pair of well-separated sets, the distance between two points from different sets can be approximated by the distance between the two sets or the distance between any pair of points in different sets. In other words, a well-separated pair decomposition can be thought of as a compressed representation to approximate the $\Theta(n^2)$ pairwise distances. Many problems that require checking of the pairwise distances can therefore be approximately solved by examining those distances between the well-separated pairs of sets. When the size of the well-separated pair decomposition is subquadratic, it often gives us more efficient algorithms than examining all the pairwise distances. Indeed, this is the intuition behind many applications of the geometric well-separated

pair decomposition. By using the same intuition, we show the application of well-separated pair decomposition in several proximity problems under the unit-disk graph metric.

Specifically, we consider the following natural proximity problems. Assume that $S_1 \subseteq S$.

- *Furthest neighbor, diameter, center.* The furthest neighbor of $p \in S_1$ is the point in S_1 that maximizes the distance to p . Related problems include computing the *diameter*, the maximum pairwise shortest distance for points in S_1 , and the *center*, the point that minimizes the maximum distance to all the other points.
- *Nearest neighbor, closest pair.* The nearest neighbor of $p \in S_1$ is the point in S_1 with the minimum distance to p . Related problems include computing the *closest pair*, the pair with minimum shortest distance, and the *bichromatic closest pair*, the pair that minimizes distance between points from two different sets.
- *Median.* The median of S is the point in S that minimizes the average (or total) distance to all the other points.
- *Stretch factor.* For a graph G defined on S , its stretch factor with respect to the unit-disk graph metric is defined to be the maximum ratio $\pi_G(p, q)/\pi(p, q)$, where π_G, π are the distances induced by G and by the unit-disk graph, respectively.

All the above problems can be solved or approximated efficiently for points in the Euclidean space. However, for the metric induced by a graph, even for planar graphs, very little is known other than solving the expensive all-pairs shortest path problem. For computing diameter, there is a simple linear time method that achieves a 2-approximation² and a 4/3-approximate algorithm with running time $O(m\sqrt{n \log n} + n^2 \log n)$ for a graph with n vertices and m edges [2]. By using the powerful tool of the well-separated pair decomposition, we are able to obtain, for all the above problems, nearly linear time algorithms for computing 2.42-approximation³ and $O(n\sqrt{n \log n}/\varepsilon^3)$ time algorithms for computing $(1 + \varepsilon)$ -approximation for any $\varepsilon > 0$. In addition, the well-separated pair decomposition can be used to obtain an $O(n \log n/\varepsilon^4)$ space distance oracle so that any $(1 + \varepsilon)$ distance query in the unit-disk graph can be answered in $O(1)$ time.

While the existence of almost linear size well-separated pair decomposition has reduced the number of pairs needed to examine when solving many proximity problems, we still need good approximation of the distances between those pairs. Our construction algorithm only produces well-separated pair decompositions without knowing an accurate approximation of the distances. For approximation algorithms, we need accurate estimation of shortest distances between $O(n \log n)$ pairs of points in the unit-disk graph. Indeed, the approximation ratio and the running time of our algorithms are dominated by the efficiency of such algorithms. Once the distance estimation has been made, the remaining computation only takes almost linear time.

For a general graph, it is unknown whether $O(n \log n)$ -pairs shortest path dis-

²Select an arbitrary node v and compute the shortest path tree rooted at v . Suppose that the furthest node from v is of distance D away. Then the diameter of the graph is no longer than $2D$, by triangular inequality.

³For a minimization problem, a quantity $\hat{\ell}$ is a c -approximation of ℓ if $\ell \leq \hat{\ell} \leq c\ell$. An object \hat{O} is a c -approximation of O with respect to a cost function f if $f(O) \leq f(\hat{O}) \leq cf(O)$. For a maximization problem, $\hat{\ell}$ is a c -approximation of ℓ if $\ell/c \leq \hat{\ell} \leq \ell$, and \hat{O} is a c -approximation of O if $f(O)/c \leq f(\hat{O}) \leq f(O)$.

tances can be computed significantly faster than all-pairs shortest path distances. For the planar graph, one can compute $O(n \log n)$ -pairs shortest path distance in $O(n\sqrt{n \log n})$ time by using separators with $O(\sqrt{n})$ size [3]. This method extends to the unit-disk graph with constant bounded density since such graphs enjoy similar separator property as the planar graphs [28, 32]. As for approximation, Thorup [35] recently discovered an algorithm for planar graphs that can answer any $(1+\varepsilon)$ -shortest distance query in $O(1/\varepsilon)$ time after almost linear time preprocessing. Unfortunately, Thorup's algorithm uses balanced shortest path separators in planar graphs which do not obviously extend to the unit-disk graphs. On the other hand, it is known that there does exist planar 2.42-spanner for a unit-disk graph [26]. By applying Thorup's algorithm to that planar spanner, we can compute 2.42-approximate shortest path distance for $O(n \log n)$ pairs in almost linear time.

Another application of well-separated pair decomposition is that we are able to obtain an almost linear size data structure to answer $(1 + \varepsilon)$ -approximate shortest path query in $O(1)$ time. Approximate distance oracles have been studied where the emphasis is often on the size of the oracles (for a survey, see [38]). For general graphs, it has been shown that it is possible to construct a $(2k - 1)$ -approximate distance oracle with size $O(kn^{1+1/k})$ [36]. It is also shown in [36] that this bound is tight for some small k 's and is conjectured to be tight for all the k 's. For planar graphs, Thorup [35] and Klein [24] have shown that there exists $(1 + \varepsilon)$ -approximate distance oracle by using almost linear space for any $\varepsilon > 0$. As mentioned, their results do not extend to the unit-disk graph. In addition, the query time of their algorithm is $O(1/\varepsilon)$. Recently, Gudmundsson et al. showed that when a geometric graph is a Euclidean spanner, there does exist an almost linear time (and therefore almost linear space) method to construct $(1 + \varepsilon)$ -approximate and $O(1)$ query time distance oracles [19]. Again, a unit-disk graph is not necessarily a Euclidean spanner with bounded stretch factor, and their technique does not extend.

3. Definitions. *Unit-disk graphs.* Denote by $d(\cdot, \cdot)$ the Euclidean metric. For a set of points S in the plane, the unit-disk graph $I(S) = (S, E)$ is defined to be the weighted graph where an edge $e = (p, q)$ is in the graph if $d(p, q) \leq 1$, and the weight of e is $d(p, q)$. Likewise, we can define the unit-ball graph for points in higher dimensions.

Metric space. Suppose that (S, π) is a metric space where S is a set of elements and π the distance function defined on $S \times S$. For any subset $S_1 \subseteq S$, the *diameter* $D_\pi(S_1)$ (or $D(S_1)$ when π is clear from the context) of S is defined to be $\max_{s_1, s_2 \in S_1} \pi(s_1, s_2)$. The *distance* $\pi(S_1, S_2)$ between two sets $S_1, S_2 \subseteq S$ is defined to be $\min_{s_1 \in S_1, s_2 \in S_2} \pi(s_1, s_2)$.

In this paper, we are interested in the *unit-disk graph metric* $\pi = \pi_{I(S)}$ induced by the unit-disk graph of a set of points S in the plane, where the distance between any two nodes is defined to be the length of the shortest path between them.

Well-separated pair decomposition. For a metric space (S, π) , two nonempty subsets $S_1, S_2 \subseteq S$ are called *c-well-separated* if $\pi(S_1, S_2) \geq c \cdot \max(D_\pi(S_1), D_\pi(S_2))$.

Following the definition in [10], for any two sets A and B , a set of pairs $\mathcal{P} = \{P_1, P_2, \dots, P_m\}$, where $P_i = (A_i, B_i)$, is called a *pair decomposition* of (A, B) (or of A if $A = B$) if

- for all the i 's, $A_i \subseteq A$ and $B_i \subseteq B$;
- $A_i \cap B_i = \emptyset$;
- for any two elements $a \in A$ and $b \in B$, there exists a unique i such that $a \in A_i$ and $b \in B_i$. We call (a, b) is *covered* by the pair (A_i, B_i) .

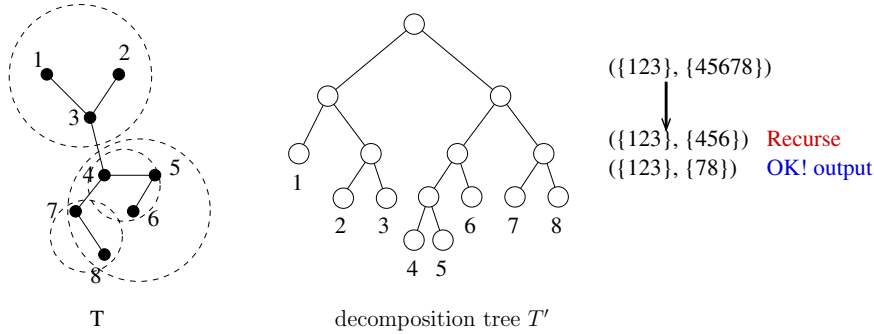


FIG. 4.1. An example of the decomposition tree.

If in addition, every pair in \mathcal{P} is c -well-separated, \mathcal{P} is called a c -well-separated pair decomposition (c -WSPD). Clearly, any metric space admits a c -WSPD with quadratic size by using the trivial family that contains all the pairwise elements.

4. Well-separated pair decomposition for unit-disk graph metric. We start with the point set with constant bounded density. Then, by combining with geometric well-separated pair decomposition, we show the extension of the result to arbitrary point sets. We will focus our discussion on points in the plane, but most results extend to higher dimensions, resulting in subquadratic size well-separated pair decomposition. We also show that our bounds in \mathbb{R}^k for $k \geq 3$ are tight.

4.1. Point sets with constant bounded density. The density α of a point set S is defined to be the maximum number of points in S covered by a unit disk. S has constant bounded density if its density is $O(1)$. We assume that the unit-disk graph on S is connected; otherwise, we can consider each connected component separately.

To construct a well-separated pair decomposition, we first compute the unit-disk graph $I(S)$ of S and then a spanning tree T of $I(S)$, where the maximum degree of T is 6. This can be done by computing the relative neighborhood graph of S [37] and keeping those edges with length at most 1. Let G be the resulting graph. It is well known that G is connected, and the degree of G is at most 6. We then compute a spanning tree of G . This step takes $O(n \log n)$ time [33]. It is also known that any n -vertex tree with maximum degree $\beta - 1$ can be divided into two parts by removing a single edge so that each subtree contain at least n/β vertices. We now recursively apply the balanced partitioning to obtain a balanced hierarchical decomposition of T (see Figure 4.1). The decomposition can be represented as a rooted binary tree T' where each node $v \in T'$ corresponds to a (connected) subtree $T(v)$ of T . The root of T' corresponds to T , and for a node $v \in T'$, v 's two children v_1, v_2 represent the two connected subtrees $T(v_1)$ and $T(v_2)$ obtained by removing an edge from $T(v)$. We denote by $S(v)$ the set of points in the subtree in $T(v)$. For a node $v \in T'$, denote by $P(v)$ the parent node of v in T' . We also use $P(S(u))$ to denote $S(P(u))$. The height of the tree T' is obviously $O(\log n)$.

Now, we describe a procedure to produce a c -WSPD of S .

For each node $v \in T'$, we pick an arbitrary point from $S(v)$ as a representative of $S(v)$ and denote it by $\sigma(S(v))$ (or $\sigma(v)$). We place in a queue the pair $(S(r), S(r))$, where r is the root of T' . We run the following process until the queue becomes empty: repeatedly remove the first element $(S(v_1), S(v_2))$ from the queue. There are two cases:

- $d(\sigma(v_1), \sigma(v_2)) \geq (c+2) \cdot \max(|S(v_1)|-1, |S(v_2)|-1)$. In this case, we include the pair to \mathcal{P} .
- $d(\sigma(v_1), \sigma(v_2)) < (c+2) \cdot \max(|S(v_1)|-1, |S(v_2)|-1)$. If $|S(v_1)| = |S(v_2)| = 1$, then it must be the case that $S(v_1)$ and $S(v_2)$ contain the same point. In this case, we simply discard the pair. Otherwise, suppose that $|S(v_1)| \geq |S(v_2)|$ and that u_1, u_2 are two children of v_1 . We add to the queue two pairs $(S(u_1), S(v_2))$ and $(S(u_2), S(v_2))$.

The above process is very similar to the collision detection algorithm in [20] except that here a pair is produced when they are c -well-separated. We now make the following claims.

LEMMA 4.1. *\mathcal{P} is a c -WSPD of S . Furthermore, each ordered pair of distinct points (p, q) is covered by exactly one pair in \mathcal{P} .*

Proof. By the construction, a pair $(S(v_1), S(v_2))$ is included in \mathcal{P} if and only if $d(\sigma(v_1), \sigma(v_2)) \geq (c+2) \cdot \max(|S(v_1)|-1, |S(v_2)|-1)$. Since for any $v \in T'$, $S(v)$ is connected, $D_\pi(S(v)) \leq |S(v)|-1$. In addition, $\pi(p, q) \geq d(p, q)$. Thus, we have that

$$\begin{aligned}
& \pi(S(v_1), S(v_2)) \\
& \geq \pi(\sigma(v_1), \sigma(v_2)) - (D_\pi(S(v_1)) + D_\pi(S(v_2))) \\
& \geq d(\sigma(v_1), \sigma(v_2)) - 2 \max(|S(v_1)|-1, |S(v_2)|-1) \\
& \geq c \cdot \max(|S(v_1)|-1, |S(v_2)|-1) \\
& \geq c \cdot \max(D_\pi(S(v_1)), D_\pi(S(v_2))).
\end{aligned}$$

That is, every pair in \mathcal{P} is a c -well-separated pair. The process clearly ends. To argue that \mathcal{P} covers all the pairs of distinct points, we observe that we begin with the pair $(S(r), S(r))$ that covers all the pairs, and each time when we split a node, the union of the pairs covered remain the same. The pairs we discard are of the form $(\{p\}, \{p\})$. Thus, all the ordered pairs of distinct points are covered by \mathcal{P} . Since the splitting produces two disjoint sets, each ordered pair is covered exactly once. \square

The following lemma shows that the sizes of two sets in the same pair do not differ too much.

LEMMA 4.2. *Each pair (A, B) that ever appears in the queue satisfies $1/\beta \leq |A|/|B| \leq \beta$.*

Proof. The proof is done by induction. Clearly, it is true for the pair $(S(r), S(r))$. Now, consider the splitting that generates the pair (A, B) . Without loss of generality, assume that we split $P(B)$, the parent node of B . By the splitting rule, we have that $|A| \leq |P(B)|$. By induction hypothesis, $|A| \geq |P(B)|/\beta \geq |B|/\beta$. Since the splitting is balanced, $|B| \geq |P(B)|/\beta \geq |A|/\beta$. Therefore $1/\beta \leq |A|/|B| \leq \beta$. \square

Now, we bound the size of \mathcal{P} .

LEMMA 4.3. *If $(A, B_i) \in \mathcal{P}$, $i = 1, \dots, m(A)$, then $B_i \cap B_j = \emptyset$, and $m(A) = O(c^2|A|)$.*

Proof. By Lemma 4.1, each pair of points can be covered only once; thus $B_i \cap B_j = \emptyset$ if both (A, B_i) and (A, B_j) are in \mathcal{P} .

If $(A, B_i) \in \mathcal{P}$, then $(P(A), P(B_i))$ is not in \mathcal{P} . So $d(\sigma(P(A)), \sigma(P(B_i))) < (c+2) \cdot \max(|P(A)|-1, |P(B_i)|-1)$. Set $R = \beta|P(A)| \leq \beta^2|A|$. If we split $P(B_i)$ to get the pair (A, B_i) , then $(A, P(B_i))$ appeared in the queue, by Lemma 4.2, we have $|P(B_i)| \leq \beta|A| \leq \beta|P(A)| = R$. If we split $P(A)$ to get the pair (A, B_i) , then $|B_i| \leq |P(A)|$, so $|P(B_i)| \leq \beta|B_i| \leq \beta|P(A)| = R$. Then,

$$d(\sigma(P(A)), \sigma(P(B_i))) < (c+2)R, D_\pi(P(B_i)) \leq R.$$

Then all the points in B_i must be inside a disk of radius $(c+3)R$ centered at $\sigma(P(A))$. Therefore we have that $|\cup_{i=1}^{m(A)} B_i| = O((c+3)^2 R^2)$ because S has constant

bounded density. By Lemma 4.2, we know that $|B_i| \geq |A|/\beta \geq |P(A)|/\beta^2$. Thus, $|B_i| \geq R/\beta^3$. Then, we have that $m(A) = O((c+3)^2 R^2/(R/\beta^3)) = O(c^2 R) = O(c^2 |A|)$. \square

LEMMA 4.4. $|\mathcal{P}| = O(c^2 n \log n)$.

Proof. Define $V_i = \{v \in T' \mid |S(v)| \in [2^i, 2^{i+1})\}$ for $0 \leq i \leq \log n$. Clearly, $|V_i| = O(n/2^i)$. Define $\Sigma_i = \{(S(v), B) \in \mathcal{P} \mid v \in V_i\}$. Denote by $m(S(v))$ the total number of pairs in which $S(v)$ is involved. By Lemma 4.3, we have that

$$\begin{aligned} |\Sigma_i| &= \sum_{v \in V_i} m(S(v)) = \sum_{v \in V_i} O(c^2 |S(v)|) \\ &= O(c^2 2^{i+1} \cdot n/2^i) = O(c^2 n). \end{aligned}$$

Thus, $|\mathcal{P}| = \sum_{i=0}^{\log n} |\Sigma_i| = O(c^2 n \log n)$. \square

Combining the above result, we now have the following theorem.

THEOREM 4.5. *For any n points with constant-bounded density in the plane and any $c \geq 1$, there exists a c -WSPD with $O(c^2 n \log n)$ pairs, which can be computed in $O(c^2 n \log n)$ time.*

Proof. Clearly, the time needed is proportional to the number of pairs that ever appear in the queue. We can represent the construction as a tree: each pair corresponds to a node in the tree, and when a pair is split, we treat those two resulting pairs as the children of the pair. Clearly, the leaves of the tree correspond to those pairs included in \mathcal{P} and the pairs discarded. All the discarded pairs have the form $(\{p\}, \{p\})$, and there are $O(n)$ such pairs. Thus, the total number of nodes in the tree is bounded by $O(|\mathcal{P}|) = O(c^2 n \log n)$. Each split costs $O(1)$. Therefore, the total computation cost is $O(c^2 n \log n)$. \square

The result can be easily extended to the point set with maximum density α .

COROLLARY 4.6. *For a point set with maximum density α , for any $c \geq 1$, a c -WSPD with $O(\alpha c^2 n \log n)$ pairs can be constructed in $O(\alpha c^2 n \log n)$ time.*

Proof. If the point set has maximum density α , Lemma 4.3 still holds if we change $m(A)$ to $O(\alpha c^2 |A|)$. Substituting the value in Lemma 4.4, we have that $|\mathcal{P}| = O(\alpha c^2 n \log n)$. The claim then follows from Theorem 4.5. \square

By a similar argument, we can extend the result to higher dimensions.

THEOREM 4.7. *Given a point set in \mathbb{R}^k , where $k \geq 3$, with constant bounded density and any constant $c \geq 1$, there exist a c -WSPD with $O(n^{2-2/k})$ pairs for the unit-ball graph metric. This bound is tight in the worst case. And the decomposition can be computed in $O(n^{2-2/k})$ time.*

Proof. We first compute a spanning tree of S with constant maximum degree β_k , a constant dependent on k only. This can be done by using the technique in [4]. We then follow the same process as described above. The upper bound follows from the same packing argument as in Lemma 4.4. Lemma 4.3 can be changed so that the number of pairs associated with a node A is $m(A) = O(|A|^{k-1})$. In addition, by Lemma 4.2, for any pair $(A, B) \in P$, $1/\beta_k \leq |A|/|B| \leq \beta_k$. Thus, $m(A) = O(n/|A|)$. Define V_i as in Lemma 4.4, $|V_i| = O(n/2^i)$. When $0 \leq i \leq \frac{1}{k} \log n$, $|\Sigma_i| = \sum_{v \in V_i} m(S(v)) = O(\sum_{v \in V_i} |S(v)|^{k-1}) = O(2^{i(k-1)} \cdot n/2^i) = O(n2^{i(k-2)})$. When $i > \frac{1}{k} \log n$, $|\Sigma_i| = \sum_{v \in V_i} m(S(v)) = O(\sum_{v \in V_i} n/|S(v)|) = O((n/2^i)^2)$. Therefore,

$$\begin{aligned} |\mathcal{P}| &= \sum_{0 \leq i \leq \frac{1}{k} \log n} n2^{i(k-2)} + \sum_{\frac{1}{k} \log n < i \leq \log n} O(n^2/2^{2i}) \\ &= O(n^{2-2/k}). \end{aligned}$$

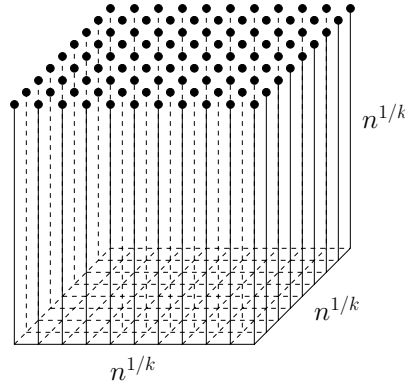


FIG. 4.2. A lower bound example of the well-separated pair decomposition for points in $k = 3$ dimensions. The shaded dots are tip points.

As for the lower bound, consider the points on the k -dimensional grid $[0, n^{1/k}) \times \dots \times [0, n^{1/k})$. Define a graph G with edges between pairs of points $(x_1, \dots, x_i, \dots, x_k)$ and $(x_1, \dots, x_i + 1, \dots, x_k)$ for $i = 1$, or $x_1 = 0$ and $i \geq 2$. A point $(n^{1/k} - 1, x_2, \dots, x_k)$ for $0 \leq x_i < n^{1/k}$ is called a *tip point*. Intuitively, G can be thought as a graph where the tip points dangle down from a $(k - 1)$ -dimensional mesh. See Figure 4.2. Clearly, we can perturb the point set so that its unit-ball graph equals G . The metric defined by G has the following property: (i) the diameter of G is $kn^{1/k}$; (ii) the distance between any two tip points is at least $2n^{1/k}$. Therefore, when $c > k/2$, a c -WSPD cannot have two tip points in the same set of a pair. Since there are $\Theta(n^{1-1/k})$ tip points, $\Omega(n^{2-2/k})$ pairs are needed, just to separate those tip points.

By the same argument as in Theorem 4.5, it is easily seen that the c -WSPD can be computed in $O(n^{2-2/k})$ time. \square

4.2. Arbitrary point sets. The packing argument fails for the unit-disk graph of point sets with unbounded density. However, we can reduce the problem to the constant density case by first clustering the points and then considering those crowded points separately by using geometric well-separated pair decompositions.

For $0 \leq \delta \leq 1$, a point p is δ -covered (or simply covered) by a point s if $d(s, p) \leq \delta$. Denote by $U(s)$ the set of points δ -covered by s . A subset $X \subseteq S$ is called a δ -cover of S if any point in S is δ -covered by some point in X . We call the points in a δ -cover X *clusterheads*. For each point in S , we assign it to the nearest clusterhead. Thus X induces a partitioning of S into sets $C(s) = S \cap \text{Vor}(s)$, where $\text{Vor}(s)$ denotes the Voronoi cell of s in X . Clearly, for any $p \in C(s)$, $d(s, p) \leq \delta$, i.e., $C(s) \subseteq U(s)$. A δ -cover is called *minimal* if no two points in X are within distance δ to each other. For any set $A \subseteq X$, denote by \hat{A} the set $\hat{A} = \cup_{s \in A} C(s)$.

To deal with an arbitrary point set S , we first compute a minimal cover X of S with an appropriately chosen δ . We then apply our results on constant-bounded density point sets to X . Note that we cannot use the unit-disk graph on X because it may not have the same connectivity as the unit-disk graph on S . Denote any two points s_1, s_2 in X *neighbors* if $d(s_1, s_2) > 1$, and there exist two points $p_1 \in C(s_1)$ and $p_2 \in C(s_2)$ such that $d(p_1, p_2) \leq 1$. We call the pair (p_1, p_2) a *bridge* between s_1 and s_2 . For each neighboring pair, we only pick one bridge arbitrarily. Let Y denote the set of all bridge points. Consider the point set $Z = X \cup Y$. Let π' denote the unit-disk graph metric on the set Z . Now, we make the claim in the next lemma.

LEMMA 4.8. X has $O(1/\delta^2)$ -density. Z can be computed in $O(n \log n/\delta^2)$ time.

Proof. Any two points s_1, s_2 in a minimal cover X are of at least distance δ away from each other. Therefore, there are $O(1/\delta^2)$ points of X inside any unit disk. So X has $O(1/\delta^2)$ density.

To compute X , we can use a greedy algorithm with the assistance of a dynamic point location data structure of unit disks [11]. The algorithm runs in $O(n \log n)$ time. To compute all the neighboring pairs, we can enumerate all the pairs (s_1, s_2) , where s_2 is inside the square centered at s_1 and with side-length $2(1 + 2\delta)$. There are $O(n/\delta^2)$ such pairs according to Lemma 4.8, and they can be computed in $O(n \log n/\delta^2)$ time by using a standard rectangular range searching data structure. Call such pairs *candidate pairs*. Clearly, only a candidate pair can possibly be a neighboring pair. To find a bridge between two clusterheads s_1, s_2 of a candidate pair, we can compute the bichromatic closest pair between two sets $C(s_1), C(s_2)$. In the plane, this can be done in $O(|U(s_1) \cup U(s_2)| \log n)$ time. Since we need only to examine each clusterhead against $O(1/\delta^2)$ clusterheads, the total computation time is bounded by $O(n \log n/\delta^2)$ [1]. \square

Now we show that π' approximates π well on the set X .

LEMMA 4.9. For any two points $p, q \in X$,

$$\pi(p, q) \leq \pi'(p, q) \leq (1 + 12\delta)\pi(p, q) + 12\delta.$$

Proof. Since $Z \subseteq S$, $\pi(p, q) \leq \pi'(p, q)$. On the other hand, assume that $p_0 p_1 \cdots p_m$, where $p_0 = p$ and $p_m = q$ is a shortest path between p and q in the unit-disk graph of S . For $0 \leq i \leq m$, suppose that s_i is the clusterhead that covers p_i . Note that $s_0 = p$ and $s_m = q$ as $p, q \in X$.

Consider two consecutive points p_i, p_{i+1} . If $s_i = s_{i+1}$, then $d(p_i, p_{i+1}) \leq 2\delta$. Otherwise, suppose that $s_i \neq s_{i+1}$. If $d(s_i, s_{i+1}) \leq 1$, then $\pi'(s_i, s_{i+1}) = d(s_i, s_{i+1}) \leq d(p_i, p_{i+1}) + 2\delta$. If $d(s_i, s_{i+1}) > 1$, then s_i, s_{i+1} must be a neighboring pair since $d(p_i, p_{i+1}) \leq 1$. In this case, it is easy to verify that $\pi'(s_i, s_{i+1}) \leq d(p_i, p_{i+1}) + 6\delta$. Thus,

$$\begin{aligned} \pi'(p, q) &\leq \sum_{i=0}^{m-1} \pi'(s_i, s_{i+1}) \\ &\leq \sum_{i=0}^{m-1} d(p_i, p_{i+1}) + 6m\delta \leq \pi(p, q) + 6m\delta. \end{aligned}$$

Since $p_0 p_1 \cdots p_m$ is a shortest path, $d(p_i, p_{i+2}) \geq 1$ for any $0 \leq i \leq m - 2$ because otherwise the path could be shortened due to triangular inequality. That is, $\pi(p, q) \geq \lfloor m/2 \rfloor > m/2 - 1$, i.e., $m < 2(\pi(p, q) + 1)$. Thus we have that $\pi'(p, q) \leq (1 + 12\delta)\pi(p, q) + 12\delta$. \square

Before we describe the construction of c -WSPD for S , we need a straightforward extension of geometric well-separated pair decomposition in [10] to two separable point sets.

LEMMA 4.10. Suppose that A and B are two point sets that can be separated by a line and have n points in total. For any constant $c \geq 1$, there exists a geometric c -well-separated pair decomposition of (A, B) with $O(n)$ pairs.

Proof. This can be done by modifying the algorithm in [10] so that the first split of the point set of $A \cup B$ is by the line that separates A and B . \square

Now, we describe a process that produces a c -WSPD of S for any $c \geq 1$. Set $\delta = 1/(2c + 4)$ and $c' = 9(c + 14)$. We first construct a minimal δ -cover X and the

set Z as described above. Next we compute a c' -well-separated pair decomposition of the clusterheads X in the unit-disk graph metric of point set Z . Specifically, we give weight 1 to points in X and 0 to bridge points. We find the spanning tree T of the unit-disk graph $I(Z)$. T has total weight $|X|$. We then recursively find balanced weighted decomposition of T : by removing an edge, each subtree has weight at least $1/\beta$ times the weight of the parent. Since X has a bounded density $O(1/\delta^2)$, the packing argument is still valid and we can compute a c' -well-separated pair decomposition for X . Suppose the decomposition obtained is $\mathcal{P} = \{P_1, P_2, \dots, P_m\}$, where $P_i = (A_i, B_i)$, $A_i \subseteq X$, $B_i \subseteq X$. We now create a set of pairs $\mathcal{P}' = \mathcal{P}'_1 \cup \mathcal{P}'_2 \cup \mathcal{P}'_3$ as follows:

1. For each $P_i \in \mathcal{P}$, if $|A_i| > 1$ or $|B_i| > 1$, we include in \mathcal{P}'_1 the pair $P'_i = (\hat{A}_i, \hat{B}_i)$. Recall that $\hat{A} = \cup_{s \in A} C(s)$.
2. If $|A_i| = |B_i| = 1$, suppose that $A_i = \{a\}$ and $B_i = \{b\}$. If $d(a, b) \geq (2c+2)\delta$, we then include in \mathcal{P}'_1 the pair $P'_i = (\hat{A}_i, \hat{B}_i)$. Otherwise, any pair of points in $\hat{A}_i \cup \hat{B}_i$ is within distance $(2c+2)\delta + 2\delta = 1$. Since $\hat{A}_i \subset \text{Vor}(a)$, and $\hat{B}_i \subset \text{Vor}(b)$, \hat{A}_i and \hat{B}_i are separable by a line. Per Lemma 4.10, we compute a geometric c -WSPD of (\hat{A}_i, \hat{B}_i) and include in \mathcal{P}'_2 all the pairs produced this way.
3. For every $s \in X$, we compute a geometric c -WSPD of $C(s)$ and include into \mathcal{P}'_3 all the pairs produced.

Now, we make the next claim.

LEMMA 4.11. \mathcal{P}' is a c -WSPD of S .

Proof. We first argue that \mathcal{P}' is a pair decomposition of S . For any pair of points $s_1, s_2 \in S$, suppose that the clusterheads covering them are s'_1 and s'_2 , respectively. If $s'_1 \neq s'_2$, then (s_1, s_2) is covered by a pair in $\mathcal{P}'_1 \cup \mathcal{P}'_2$. Otherwise, it is covered by a pair in \mathcal{P}'_3 . It is also easily verified that each ordered pair is covered exactly once.

Now, we show that all the pairs in \mathcal{P}' are c -well-separated with respect to the unit-disk graph metric. Since $\delta = 1/(2c+4)$, for all the pairs in \mathcal{P}'_2 , the Euclidean distance between any two points in $\hat{A}_i \cup \hat{B}_i$ is at most $(2c+4)\delta = 1$. Therefore, the unit-disk graph on the subset $\hat{A}_i \cup \hat{B}_i$ is a complete graph, i.e., every pair in \mathcal{P}'_2 is c -well-separated under the unit-disk graph metric. The same argument applies to \mathcal{P}'_3 as the distance between two points in $C(s)$ is at most $2\delta \leq 1$.

Now, consider a pair $(\hat{A}_i, \hat{B}_i) \in \mathcal{P}'_1$. We distinguish two cases:

1. When $|A_i| = |B_i| = 1$. Then we must have $\pi(A_i, B_i) \geq (2c+2)\delta$ according to the construction rule, and thus

$$\pi(\hat{A}_i, \hat{B}_i) \geq \pi(A_i, B_i) - 2\delta \geq 2c\delta = c/(c+2).$$

On the other hand, $D(\hat{A}_i), D(\hat{B}_i) \leq 2\delta = 1/(c+2)$. Therefore, (\hat{A}_i, \hat{B}_i) is c -well-separated.

2. When $|A_i| > 1$ or $|B_i| > 1$. In what follows, we use D and D' to denote $D_\pi, D_{\pi'}$, respectively. Clearly,

$$\pi(\hat{A}_i, \hat{B}_i) \geq \pi(A_i, B_i) - 2\delta, \text{ and } D(\hat{A}) \leq D(A) + 2\delta.$$

Since either A_i or B_i contains at least two clusterheads, it must be true that $\max(D(A_i), D(B_i)) \geq \delta$, as the distance between two clusterheads is at least δ . So, $\max(D(\hat{A}_i), D(\hat{B}_i)) \geq \delta$, and $\max(D(\hat{A}_i), D(\hat{B}_i)) \leq \max(D(A_i), D(B_i)) + 2\delta \leq 3 \max(D(A_i), D(B_i))$.

As A_i, B_i are c' -well-separated under π' , $\pi'(A_i, B_i) \geq c' \cdot \max(D'(A_i), D'(B_i))$.

Therefore,

$$\begin{aligned}
\pi(\hat{A}_i, \hat{B}_i) &\geq \pi(A_i, B_i) - 2\delta \\
&\geq (\pi'(A_i, B_i) - 12\delta)/(1 + 12\delta) - 2\delta \\
&\quad \text{by Lemma 4.9} \\
&\geq c'/(1 + 12\delta) \cdot \max(D'(A_i), D'(B_i)) - 14\delta \\
&\geq c'/(1 + 12\delta) \cdot \max(D(A_i), D(B_i)) - 14\delta \\
&\geq (c'/(3(1 + 12\delta)) - 14) \cdot \max(\hat{D}(A_i), \hat{D}(B_i)) \\
&\geq c \max(D(\hat{A}_i), D(\hat{B}_i)). \\
&\quad \text{by } c \geq 1, \delta = 1/(2c + 4), \text{ and } c' = 9(c + 14).
\end{aligned}$$

In both cases, \hat{A}_i, \hat{B}_i are c -well-separated, i.e., all the pairs in \mathcal{P}'_1 are c -well-separated. \square

Now, we make the claim in the next theorem.

THEOREM 4.12. *For any set S of n points in the plane and any $c \geq 1$, there exists a c -WSPD \mathcal{P} of S under the unit-disk graph metric where \mathcal{P} contains $O(c^4 n \log n)$ pairs and can be computed in $O(c^4 n \log n)$ time.*

Proof. By combining Corollary 4.6 and Lemma 4.8, we have that $|\mathcal{P}'_1| \leq |\mathcal{P}| = O(c^2 n \log n / \delta^2) = O(c^4 n \log n)$. If $|A_i| = 1$, then the number of pairs $(A_i, B_i) \in \mathcal{P}'_2$ where $|B_i| = 1$ is bounded by $O(1/\delta^2) = O(c^2)$. Since the size of the geometric well-separated pair decomposition is linear in terms of the number of points [10], $|\mathcal{P}'_2| = O(c^2 n)$. Clearly, $|\mathcal{P}'_3| = O(n)$. When we sum the sizes, we have that $|\mathcal{P}'| = O(c^4 n \log n)$.

By Theorem 4.5 and Lemma 4.8, it is easy to see that the total time needed is $O(c^4 n \log n)$. \square

Similarly, in higher dimensions, we have the next corollary.

COROLLARY 4.13. *For any set S of n points in \mathbb{R}^k , for $k \geq 3$, and for any constant $c \geq 1$, there exists a c -WSPD \mathcal{P} of S under the unit-ball graph metric where \mathcal{P} contains $O(n^{2-2/k})$ pairs and can be constructed in $O(n^{4/3} \text{polylog } n)$ time for $k = 3$ and in $O(n^{2-2/k})$ time for $k \geq 4$.*

Proof. For simplicity of computation, we use boxes instead of balls to find clusterheads with constant bounded density. A point p is covered by a point s if p is inside the box with size 2δ centered at s . Finding the minimal cover can be done by using a dynamic rectilinear range search tree in k -dimensions [11]. The running time is $O(n \text{polylog } n)$. Notice that every point can be covered by at most a constant number of clusterheads; thus we can find the nearest clusterhead for every point in linear time in total. To find a bridge between two clusterheads s_1, s_2 , we compute the bichromatic closest pair between two sets $C(s_1), C(s_2)$. Let $m_1 = |C(s_1)|$ and $m_2 = |C(s_2)|$. According to [1], when $k = 3$, it takes $O((m_1 m_2)^{2/3} \text{polylog } n)$ time, and when $k = 4$, it is

$$\begin{aligned}
&O((m_1 m_2)^{1-1/(\lceil k/2 \rceil + 1)+\epsilon} + m_1 \log m_2 + m_2 \log m_1) \\
&= O((m_1 m_2)^{1-1/k} + m_1 \log m_2 + m_2 \log m_1).
\end{aligned}$$

Since each set is involved only in $O(1)$ bichromatic closest pair computation, the total time is $O(n^{4/3} \text{polylog } n)$ when $k = 3$ and $O(n^{2-2/k})$ for $k \geq 4$. Computing the WSPD on the clusterheads takes $O(n^{2-2/k})$ time, according to Theorem 4.5. \square

4.3. Estimating distance between pairs. In the above, we showed how to construct well-separated pair decomposition for unit-disk and unit-ball graphs. As mentioned in the introduction, to apply WSPD in solving proximity problems in

the unit-disk graphs, we first need to estimate the shortest path distances between $O(n \log n)$ pairs of the WSPD. Note that in our construction for the point sets with constant bounded density, we use Euclidean distance as a lower bound for the unit-disk graph distance and the size of the point set as an upper bound for the diameter. While these approximations are sufficient for bounding the size of WSPDs, they are too coarse for obtaining good approximation. Recall that $\sigma(A)$ is an (arbitrary) point picked from a set A . For a c -well-separated pair (A, B) , we can use the estimated distance $\hat{\pi}(\sigma(A), \sigma(B))$ to approximate all the pairwise distances between points in A and points in B . In this section, we show several trade-offs for measuring the distance between m pairs of points in the unit-disk graph.

Denote by $\tau(n, c, m)$ the time needed to compute m -pairs c -approximate distance in a unit disk graph. In what follows, we set $c_0 = 2.42 > \frac{4\sqrt{3}}{9}\pi$ and c_1 a number slightly smaller than c_0 but greater than $\frac{4\sqrt{3}}{9}\pi$.

LEMMA 4.14.

1. $\tau(n, c_1, m) = O(n \log^3 n + m)$.
2. $\tau(n, 1 + \varepsilon, m) = O(n^2/(\varepsilon r) + mr/\varepsilon)$, for any $1 \leq r \leq n$.

Proof. 1. We first construct a planar $\frac{4\sqrt{3}}{9}\pi$ -spanner of the unit disk graph. Such spanner exists and can be computed in $O(n \log n)$ time [26]. Now, we apply Thorup's construction of $(1 + \varepsilon)$ -approximate distance oracle [35] to that planar spanner, for a sufficiently small constant $\varepsilon > 0$. The bound follows immediately from the preprocessing and query time bounds of Thorup's algorithm.

2. We again cluster the points and consider the set of clusterheads, X . Suppose that we have constructed a $(1 + \varepsilon/2)$ -approximate shortest distance oracle for X . For two query points q_1, q_2 , if $d(q_1, q_2) \leq 1$, we return $d(q_1, q_2)$. Otherwise, we find the clusterheads s_1, s_2 that cover q_1 and q_2 , respectively, and return $\hat{\pi}(q_1, q_2) = \pi'(s_1, s_2) + 2\delta$ as an approximation of $\pi(q_1, q_2)$. It is easily verified that $\hat{\pi}(q_1, q_2)$ is a $(1 + \varepsilon)$ -approximation for $\delta = O(\varepsilon)$. The density of X is $O(1/\delta^2) = O(1/\varepsilon^2)$. The graph formed by connecting neighboring pairs in X is an $O(1/\varepsilon^2)$ -overlap graph as defined by Miller, Teng, and Vavasis [28] and therefore admits a balanced separator with size $O(\sqrt{n}/\varepsilon)$. Furthermore, it can be computed in deterministic linear time by the method of Eppstein, Miller, and Teng [13].

Now, it is easy to extend the shortest distance algorithm for planar graphs by Arikati et al. [3] to the above geometric graph on X . By using the same technique, we can obtain a trade-off with $O(n^2/(\varepsilon r))$ preprocessing time and $O(r/\varepsilon)$ query time for any $1 \leq r \leq \sqrt{n}$. \square

5. Applications. In this section, we show the application of the well-separated pair decomposition in obtaining efficient algorithms for approximating the furthest neighbor (diameter, center), nearest neighbor (closest pair), median, and stretch factor, all under the unit-disk graph metric. Since the running time of the algorithms for computing c_0 -approximate and $(1 + \varepsilon)$ -approximate distance are different, we will be describing the bounds for both approximations (recall that $c_0 = 2.42$). Roughly speaking, our algorithms for computing c_0 -approximation is about linear and for computing $(1 + \varepsilon)$ -approximation is about $O(n\sqrt{n})$, dominated by the distance estimation.

We should note that for the problems of computing diameter and center, there is a simple linear time method to achieve 2-approximation. It is therefore not interesting to present algorithms to obtain c_0 -approximation for those problems, with $c_0 = 2.42$. For the other problems, it is still interesting, as we are not aware of any algorithms that achieve comparable approximation ratio in subquadratic time, even for planar graphs.

We need first to describe the well-separated pair decomposition we will be using. In what follows, we also include the time for measuring the distances between pairs into the construction time. For c_0 -approximation, $c_0 = 2.42$, we construct a c -well-separated pair decomposition \mathcal{P}_1 for sufficiently large constant c and, for each pair (A, B) in the WSPD, compute c_1 -approximate distance $\hat{\pi}_1(A, B)$ between $\sigma(A)$ and $\sigma(B)$ according to Lemma 4.14.1. For $(1 + \varepsilon)$ -approximation, we compute a c -well-separated pair decomposition \mathcal{P}_2 for $c = O(1/\varepsilon)$ and, for each pair (A, B) , compute the $(1 + \varepsilon/2)$ -approximate distance $\hat{\pi}_2(A, B)$ between $(\sigma(A), \sigma(B))$ by Lemma 4.14.2 and by setting $r = \varepsilon^2 \sqrt{n/\log n}$. The following is immediate.

LEMMA 5.1. *\mathcal{P}_1 contains $O(n \log n)$ pairs and can be computed in $O(n \log^3 n)$ time. \mathcal{P}_2 contains $O(n \log n/\varepsilon^4)$ pairs and can be computed in $O(n\sqrt{n \log n}/\varepsilon^3)$ time. For any pair of points (p, q) , suppose that its covering pair in \mathcal{P}_1 (\mathcal{P}_2) is (A, B) ; then $\hat{\pi}_1(A, B)$ ($\hat{\pi}_2(A, B)$) is a c_0 -approximation ($(1 + \varepsilon)$ -approximation) of $\pi(p, q)$, $c_0 = 2.42$.*

In the process of producing a well-separated pair decomposition, we constructed several trees, the balanced hierarchical decomposition tree for constant bounded density points and the fair split trees for geometric well-separated pair decomposition [10]. For simplicity of presentation, we treat them as a single tree T'_1 and T'_2 , for \mathcal{P}_1 and \mathcal{P}_2 , respectively, by joining the trees created in the geometric well-separated pair decomposition to the clusterheads appropriately. In what follows, $\mathcal{P}, T', \hat{\pi}$ mean that they could be either case.

5.1. $(1 + \varepsilon)$ -distance oracle. Although \mathcal{P}_2 takes time $O(n\sqrt{n \log n}/\varepsilon^3)$, the space needed is only $O(n \log n/\varepsilon^4)$. We can use \mathcal{P}_2 to answer $(1 + \varepsilon)$ -approximate distance query between any two points (p_1, p_2) by first locating the pair (A, B) that covers (p, q) and returning $\hat{\pi}(A, B)$. The query time is the time needed to discover a pair in \mathcal{P}_2 that covers the query pair. We show that this can be done in $O(1)$ time by using the properties of WSPD.

COROLLARY 5.2. *For a unit-disk graph on n points and for any $\varepsilon > 0$, we can preprocess it into a data structure with $O(n \log n/\varepsilon^4)$ size so that for any query pair, a $(1 + \varepsilon)$ -approximate distance can be answered in $O(1)$ time.*

Proof. It suffices to prove for constant-bounded density point sets. We store all the pairs in \mathcal{P} in a hash table indexed by the pairs. We will show that for each query pair (p, q) , we can find $O(1)$ candidate pairs that are guaranteed to contain the pair in \mathcal{P} that covers (p, q) . Then, we simply query the hash table using those candidate pairs and discover the one that does cover (p, q) .

We modify our construction in section 4.1 so that we are more careful on deciding when to include a pair in \mathcal{P} . We use a c_1 -approximate distance oracle as constructed in Lemma 4.14.1. When producing \mathcal{P} , we include a pair in \mathcal{P} if $\hat{\pi}(A, B) > (cc_1 + 2) \max(|A| - 1, |B| - 1)$. Then there is a constant $c_2 > 0$ such that for any $c \geq 2$ and any pair $(A, B) \in \mathcal{P}$, $cc_1 s \leq \pi(A, B) \leq cc_2 s$, where $s = \max(|A| - 1, |B| - 1)$.

Now, to answer a query (p, q) , we first use the c_1 -approximate distance oracle to compute an approximation ℓ of $\pi(p, q)$, i.e., $\pi(p, q) \leq \ell \leq c_1 \pi(p, q)$. Suppose that $(A, B) \in \mathcal{P}$ is the pair that covers (p, q) . Without loss of generality, let us assume that $|A| \geq |B|$, i.e., $s = |A| - 1$. Then we have

$$s \leq \pi(A, B)/(cc_1) \leq \pi(p, q)/(cc_1) \leq \ell/(cc_1).$$

On the other hand, $s \geq \pi(A, B)/(cc_2) \geq (\pi(p, q) - 2s)/(cc_2)$. That is, $s \geq \pi(p, q)/(cc_2 + 2) \geq \ell/(c_1(cc_2 + 2))$.

Set $\hat{\ell} = \ell/(cc_1)$. Then, for (A, B) to cover (p, q) , A has to be an ancestor of p in T' , and the size of A is sandwiched by $\hat{\ell}/(c_2 + 1)$ and $\hat{\ell}$. Notice that c_1, c_2 are constants independent of c . There are only $O(1)$ such nodes in T' . Similarly, there are only $O(1)$ such B 's. We now form $O(1)$ candidate pairs by joining every pair. Clearly, this can be done in $O(1)$ time. \square

5.2. Furthest neighbor. Suppose that $S_1 \subseteq S$. For any p , define the (relative) furthest neighbor of p to be $\xi(p) = \arg \max_{q \in S_1} \pi(p, q)$ in S_1 . Then the diameter of S_1 is $D(S_1) = \max_{p \in S_1} \pi(p, \xi(p))$. The center of S_1 is the point that minimizes the maximum distance to the other points, i.e., $\arg \min_{p \in S_1} \pi(p, \xi(p))$. Therefore, once we compute approximate furthest neighbors for all the p , we also obtain approximate diameter and center.

Consider any WSPD. To compute the furthest neighbor of S_1 , we traverse the balanced hierarchical decomposition tree T' and mark all the nodes $v \in T'$, where $S(v) \cap S_1 \neq \emptyset$. This can be done in $O(n)$ time in a postorder visit of the tree. A pair $P = (S(u), S(v))$ is called *marked* if both u and v are marked. Let

$$R_1(u) = \max\{\hat{\pi}(S(u), B) \mid (S(u), B) \text{ is marked}\}$$

and 0 if there is no such pair. With each node u , we also record $\ell(u)$, the node that achieves $R_1(u)$.

For any $p \in S_1$, consider the path P in T' from p to the root. Suppose that u is the node that maximizes $R_1(u)$ among all the nodes on P . Now, we pick any point, say, q , from $S(\ell(u)) \cap S_1$ (since $\ell(u)$ is marked, $S(\ell(u)) \cap S_1 \neq \emptyset$) and claim that it is an approximate furthest neighbor with the approximation ratio 2.42, if the above process is applied to \mathcal{P}_1 , or $1 + \varepsilon$, if applied to \mathcal{P}_2 . For correctness, consider the (marked) pair in \mathcal{P} that covers $(p, \xi(p))$. Suppose it is $(S(u), S(v))$. Then $R_1(u) \geq \hat{\pi}(S(u), S(v))$. Since the pairs are well-separated, it is easy to see that q is an approximate furthest neighbor of p with the approximation ratio determined by the WSPD we use. After we have computed the approximate furthest neighbor, it is simple to compute the diameter and the center. Therefore, we have the next corollary.

COROLLARY 5.3. *For any set S of n points in the plane and any $S_1 \subseteq S$, we can compute*

- c_0 -approximate furthest neighbor for all the points in S_1 in $O(n \log^3 n)$ time, $c_0 = 2.42$; and
- $(1 + \varepsilon)$ -approximation, for any $\varepsilon > 0$, of the furthest neighbor, the diameter of S_1 , and the center of S_1 in $O(n\sqrt{n} \log n / \varepsilon^3)$ time.

Remark. We did not list c_0 -approximation ($c_0 = 2.42$) for the diameter and the center because there is a simple linear time 2-approximate algorithm.

5.3. Nearest neighbor, closest pair. Computing the nearest neighbor or closest pair in S under the unit-disk graph metric is trivial—it is the same as under the Euclidean metric as long as the graph is connected. However, the problem becomes harder if we restrict our attention to a subset $S_1 \subseteq S$, i.e., computing the nearest neighbor in S_1 for each point in S_1 or computing the closest pair between points in S_1 . For any two sets S_1, S_2 , we can also define the bichromatic closest pair to be $\arg \min_{p \in S_1, q \in S_2} \pi(S_1, S_2)$.

By using the same technique as in the previous section, we are able to show the next corollary.

COROLLARY 5.4. *For any set S of n points in the plane, and any $S_1, S_2 \subseteq S$, we can compute*

- c_0 -approximation ($c_0 = 2.42$) for the nearest neighbor for all the points in S_1 , the closest pair in S_1 , the bichromatic closest pair of S_1, S_2 , in time $O(n \log^3 n)$; and
- $(1 + \varepsilon)$ -approximation for the same problems in time $O(n\sqrt{n \log n}/\varepsilon^3)$.

Remark. We should note that by applying the technique in [10], we can actually enumerate a set of $O(n)$ pairs of points that is guaranteed to include the closest pair. However, since our distance oracle is approximate, we can compute only the approximate closest pair, unlike in the geometric case.

5.4. Median. Similar to the definition of center, median is defined to be the point that minimizes the average (or total) distance to all the other points. Let $\rho(p) = \sum_{q \in S_1} \pi(p, q)$. Then the median of S_1 is the point that minimizes $\rho(p)$.

By using a similar technique, we can show the following.

COROLLARY 5.5. *For any planar point set S with n points and $S_1 \subseteq S$, a c_0 -approximate median ($c_0 = 2.42$) of S_1 can be computed in $O(n \log^3 n)$ time, and for any $\varepsilon > 0$, a $(1 + \varepsilon)$ -approximation can be computed in $O(n\sqrt{n \log n}/\varepsilon^3)$ time.*

Proof. Computing approximate median is similar to computing the furthest neighbor. The only difference is that instead of computing $R_1(u)$, we compute

$$R_2(u) = \sum_{(S(u), B) \in \mathcal{P}} \hat{\pi}(S(u), B) \cdot |B|,$$

and then for each point p and the path P from p to the root, compute $\hat{\rho}(p) = \sum_{u \in P} R_2(u)/(n - 1)$, as an approximation of $\rho(p)$. The correctness is guaranteed by the property of pair decomposition that every pair of points is covered by a unique pair in the decomposition. Again, we pick the point with the minimum $\hat{\rho}(p)$ to be the approximate median. The approximation ratio and running time bounds follow immediately. \square

5.5. Stretch factor. For a graph G defined on S , the *stretch factor* of G with respect to π is defined as $\max_{p, q \in S} \pi_G(p, q)/\pi(p, q)$. Narasimhan and Smid [29] gave an algorithm to approximate the stretch factor of a geometric graph to the Euclidean metric using the geometric well-separated pair decomposition. By following the same argument we can approximate the stretch factor of an arbitrary graph G with respect to the unit-disk graph metric. Again, we consider the well-separated pair decomposition \mathcal{P} . For each pair $(A, B) \in \mathcal{P}$, we pick any pair of points (p, q) , where $p \in A$ and $q \in B$, and compute the approximate shortest path $\hat{\pi}_G(p, q)$ in G and $\hat{\pi}(p, q)$ in I . The maximum ratio of $\hat{\pi}_G(p, q)/\hat{\pi}(p, q)$ over all pairs in \mathcal{P} is an approximation to the stretch factor by the same argument in [29].

COROLLARY 5.6. *For any graph G on S , we can compute an $O(1)$ -approximate stretch factor of G in time $O(\tau'_1(n \log n))$, where $\tau'_1(m)$ is the time to compute m $O(1)$ -approximate shortest path queries in G . In particular, if G is a subgraph of I , an $O(1)$ -approximate can be computed in time $O(n \log^3 n)$. Similarly, we can compute for any $\varepsilon > 0$, a $(1 + \varepsilon)$ -approximate stretch factor of G in time $O(\tau'_2(n \log n/\varepsilon^4) + n\sqrt{n \log n}/\varepsilon^3)$, where $\tau'_2(m)$ is the time to compute m $(1 + \varepsilon)$ -approximate shortest path queries in G . When G is a subgraph of I , this can be done in $O(n\sqrt{n \log n}/\varepsilon^3)$ time.*

6. Extensions. There are several direct extensions of our techniques. Here, we outline the extension to the intersection graph of disks with bounded radii ratio and to the unweighted unit-disk graph.

6.1. Intersection graphs of disks with bounded radii ratio. When the sizes of the disks are not uniform, it is generally not possible to obtain subquadratic well-separated pair decomposition of the metric induced by the intersection graph. This can be shown by the example where there is a big disk and $n - 1$ pairwise disjoint small disks intersecting it. Indeed, the intersection graph of this example is a tree with one internal node and $n - 1$ leaves.

However, if the ratio between the radii of any two disks (or balls) is upper bounded by a constant, then the packing property (Lemma 4.3) still holds. We can obtain similar results for the intersection graph of disks (or balls in high dimensions) with bounded radii ratio.

6.2. Unweighted unit-disk graphs. In the previous part, we considered only weighted unit-disk graphs. There are applications in which we need the unweighted unit-disk graph. The results for point set with constant bounded density can be directly extended to unweighted unit-disk graphs. If the density is unbounded, then it is impossible to obtain a subquadratic size well-separated pair decomposition as shown by the example of the unweighted complete graph. But again for the applications, we can apply the clustering technique to reduce it to the problem for point sets with constant unbounded density. The clustering increases the approximation ratio by a multiplicative factor of 3 [17]. Thus in near linear time we can compute $3c_0$ -approximation ($c_0 = 2.42$) for the following problems: the furthest neighbor, nearest neighbor, closest pair, bichromatic closest pair, median, and stretch factor, all with respect to the unweighted unit-disk graph metric. Again we didn't list the problems of computing diameter and center because there are trivial 2-approximate algorithms.

Furthermore, we can get a better multiplicative approximation factor by permitting an additive error as shown in the following. For the unweighted unit-disk graph $I(S)$ on point set S , we cluster the points by finding a minimal 1-cover X of S ; i.e., any two clusterheads $c_1, c_2 \in X$ must be distance at least 1 away, and any point is covered by at least one clusterhead. We also assign a unique clusterhead $c(p)$ to every node p in S , as before. For any two clusterheads in X within distance 3, if there exists a path with no more than three hops to connect them, we select such two nodes as a bridge. Define Z to be the union of centers X and bridge nodes Y . Again, the shortest path metric in the unweighted unit-disk graph $I(Z)$ is denoted by π' , to be distinguished by the metric π in $I(S)$. It is easy to see that Z has constant bounded density. So we build the c -well-separated pair decomposition \mathcal{P}' on Z . For each pair $(A', B') \in \mathcal{P}'$, we build a pair (A, B) , where $A = \bigcup_{c(p) \in A'} p$, $B = \bigcup_{c(q) \in B'} q$. The collection of the pairs is denoted by \mathcal{P} .

LEMMA 6.1. For $p, q \in S$,

1. $\pi(p, q) \leq \pi'(p, q)$;
2. $\pi'(p, q) \leq 3\pi(p, q) + 2$, if p, q are clusterheads, then $\pi'(p, q) \leq 3\pi(p, q)$.

Proof. The first claim is because $I(Z)$ is a subgraph of $I(S)$. The second one is proved in [17]. \square

Set $c = 6/\varepsilon$; we have the next lemma.

LEMMA 6.2. For any two pairs of points $(p_1, q_1), (p_2, q_2) \in (A, B)$, where $(A, B) \in \mathcal{P}$, $\pi(p_1, q_1) \leq (1 + \varepsilon)\pi(p_2, q_2) + (4 + 2\varepsilon)$.

Proof. Take the centers of p_2, q_2 , $c(p_2) \in A'$, $c(q_2) \in B'$. We can see that $\pi(p_1, c(p_2)) \leq 1 + \pi(c(p_1), c(p_2)) \leq 1 + \pi'(c(p_1), c(p_2)) \leq 1 + D'(A')$, where D' denote the diameter in metric π' . Since (A', B') is c -well-separated, we have $\pi'(A', B') \geq c \cdot \max(D'(A'), D'(B'))$. So $\pi'(c(p_2), c(q_2)) \geq c \cdot \max(D'(A'), D'(B'))$. $\pi(c(p_2), c(q_2)) \leq$

$\pi(p_2, q_2) + 2$. Combining Lemma 6.1 and all these, we have

$$\begin{aligned}
\pi(p_1, q_1) &\leq \pi(p_1, c(p_2)) + \pi(c(p_2), c(q_2)) + \pi(c(q_2), q_1) \\
&\leq 1 + D'(A') + \pi(c(p_2), c(q_2)) + 1 + D'(B') \\
&\leq \pi(c(p_2), c(q_2)) + 2 + (2/c)\pi'(c(p_2), c(q_2)) \\
&\leq \pi(c(p_2), c(q_2)) + 2 + (2/c)(3\pi(c(p_2), c(q_2))) \\
&\leq (1 + 6/c)\pi(c(p_2), c(q_2)) + 2 \\
&\leq (1 + 6/c)\pi(p_2, q_2) + (4 + 12/c) \\
&= (1 + \varepsilon)\pi(p_2, q_2) + (4 + 2\varepsilon). \quad \square
\end{aligned}$$

THEOREM 6.3. *For an unweighted unit-disk graph $I(S)$ on a point set S and any $1 > \varepsilon > 0$, we can find in time $O(n\sqrt{n \log n}/\varepsilon^3)$ a data structure of size $O(n \log n/\varepsilon^4)$ such that for any pair of points p, q with distance $\pi(p, q) > 2$, we can return a value x in $O(1)$ time such that*

$$\frac{1}{1 + \varepsilon}(x - 4 - 2\varepsilon) \leq \pi(p, q) \leq (1 + \varepsilon)x + 4 + 2\varepsilon.$$

Proof. For each pair (A', B') , we take an arbitrary pair of points a_0, b_0 from A', B' , respectively, and compute the distance $\pi(a_0, b_0)$. By using the same idea as in Lemma 4.14, we can show that the total amount of time is $O(n\sqrt{n \log n}/\varepsilon^3)$. Note that any pair of points p, q with distance $\pi(p, q) > 2$ must be in different clusters. We find the pair (A', B') that includes the pair of points $(c(p), c(q))$ in $O(1)$ time. (Notice that there exists a constant-spanner for the unweighted unit-disk graph on point set with constant density [17].) Therefore we take $x = \pi(a_0, b_0)$, where (a_0, b_0) is the representative pair of (A', B') . The theorem then follows from Lemma 6.2. \square

Similarly, this gives us $O(n\sqrt{n \log n}/\varepsilon^3)$ -time algorithms for finding approximate solutions to the following problems: the furthest neighbor, nearest neighbor, closest pair, bichromatic closest pair, median, diameter, center, and stretch factor, all with respect to the unweighted unit-disk graph metric.

7. Conclusion. In this paper, we extend the well-separated pair decomposition, originally developed in the Euclidean metric, to the unit-disk and unit-ball graph metrics. This allows us to obtain almost linear time 2.42-approximate and subquadratic time $(1 + \varepsilon)$ -approximate algorithms for several proximity problems where no efficient methods were previously known. The combinatorial bounds in \mathbb{R}^k for $k \geq 3$ are also tight.

The most notable open problem is the gap between $\Omega(n)$ and $O(n \log n)$ on the number of pairs needed in the plane. Also, the time bound for $(1 + \varepsilon)$ -approximation is still about $\tilde{O}(n\sqrt{n})$ because of the lack of efficient method for computing $(1 + \varepsilon)$ -approximate shortest distance between $O(n)$ pairs of points. Any improvement to the algorithm for that problem will immediately lead to improvement to all the $(1 + \varepsilon)$ -approximate algorithms presented in this paper.

REFERENCES

- [1] P. K. AGARWAL, H. EDELSBRUNNER, O. SCHWARZKOPF, AND E. WELZL, *Euclidean minimum spanning trees and bichromatic closest pairs*, Discrete Comput. Geom., 6 (1991), pp. 407–422.
- [2] D. AINGWORTH, C. CHEKURI, AND R. MOTWANI, *Fast estimation of diameter and shortest paths (without matrix multiplication)*, in Proceedings of the 7th ACM-SIAM Symposium on Discrete Algorithms, SIAM, Philadelphia, 1996, pp. 547–553.

- [3] S. R. ARIKATI, D. Z. CHEN, L. P. CHEW, G. DAS, M. H. M. SMID, AND C. D. ZAROLIAGIS, *Planar spanners and approximate shortest path queries among obstacles in the plane*, in Proceedings of the 4th Annual European Symposium on Algorithms, J. Díaz and M. Serna, eds., 1996, pp. 514–528.
- [4] S. ARYA, G. DAS, D. M. MOUNT, J. S. SALOWE, AND M. SMID, *Euclidean spanners: Short, thin, and lanky*, in Proceedings of the 27th ACM Symposium on Theory of Computing, 1995, pp. 489–498.
- [5] S. ARYA, D. M. MOUNT, AND M. SMID, *Randomized and deterministic algorithms for geometric spanners of small diameter*, in Proceedings of the 35th IEEE Symposium on Foundations of Computer Science, 1994, pp. 703–712.
- [6] H. BREU AND D. G. KIRKPATRICK, *Unit disk graph recognition is NP-hard*, Computational Geom., 9 (1998), pp. 3–24.
- [7] CALLAHAN AND KOSARAJU, *Faster algorithms for some geometric graph problems in higher dimensions*, in Proceedings of the 4th ACM-SIAM Symposium on Discrete Algorithms, SIAM, Philadelphia, 1993, pp. 291–300.
- [8] P. B. CALLAHAN, *Optimal parallel all-nearest-neighbors using the well-separated pair decomposition*, in Proceedings of the 34th IEEE Symposium on Foundations of Computer Science, 1993, pp. 332–340.
- [9] P. B. CALLAHAN AND S. R. KOSARAJU, *Algorithms for dynamic closest-pair and n -body potential fields*, in Proceedings of the 6th ACM-SIAM Symposium on Discrete Algorithms, SIAM, Philadelphia, 1995, pp. 263–272.
- [10] P. B. CALLAHAN AND S. R. KOSARAJU, *A decomposition of multidimensional point sets with applications to k -nearest-neighbors and n -body potential fields*, J. ACM, 42 (1995), pp. 67–90.
- [11] Y.-J. CHIANG AND R. TAMASSIA, *Dynamic algorithms in computational geometry*, Proc. IEEE, 80 (1992), pp. 1412–1434.
- [12] B. N. CLARK, C. J. COLBOURN, AND D. S. JOHNSON, *Unit disk graphs*, Discrete Math., 86 (1990), pp. 165–177.
- [13] D. EPPSTEIN, G. L. MILLER, AND S.-H. TENG, *A deterministic linear time algorithm for geometric separators and its applications*, in Proceedings of the 9th Annual ACM Symposium on Computing in Geometry, 1993, pp. 99–108.
- [14] J. ERICKSON, *Dense point sets have sparse Delaunay triangulations*, in Proceedings of the 13th ACM-SIAM Symposium on Discrete Algorithms, SIAM, Philadelphia, 2002, pp. 125–134.
- [15] T. ERLEBACH, K. JANSEN, AND E. SEIDEL, *Polynomial-time approximation schemes for geometric graphs*, in Proceedings of the 12th ACM-SIAM Symposium on Discrete Algorithms, 2001, pp. 671–679.
- [16] B. FISCHL, M. SERENO, AND A. DALE, *Cortical surface-based analysis II: Inflation, flattening, and a surface-based coordinate system*, NeuroImage, 9 (1999), pp. 195–207.
- [17] J. GAO, L. J. GUIBAS, J. HERSHBERGER, L. ZHANG, AND A. ZHU, *Geometric spanners for routing in mobile networks*, IEEE J. Selected Areas Commun. Wireless Ad Hoc Networks, 23 (2005), pp. 174–185.
- [18] S. GOVINDARAJAN, T. LUKOVSKI, A. MAHESHWARI, AND N. ZEH, *I/O efficient well-separated pair decomposition and its applications*, in Proceedings of the 8th European Symposium on Algorithms, 2000, pp. 220–231.
- [19] J. GUDMUNDSSON, C. LEVCOPOULOS, G. NARASIMHAN, AND M. SMID, *Approximate distance oracles for geometric graphs*, in Proceedings of the 13th ACM-SIAM Symposium on Discrete Algorithms, SIAM, Philadelphia, 2002, pp. 828–837.
- [20] L. GUIBAS, A. NGUYEN, D. RUSSEL, AND L. ZHANG, *Collision detection for deforming necklaces*, in Proceedings of the 18th ACM Symposium on Computational Geometry, 2002, pp. 33–42.
- [21] W. K. HALE, *Frequency assignment: Theory and applications*, Proc. IEEE, 68 (1980), pp. 1497–1513.
- [22] H. B. H. III, M. V. MARATHE, V. RADHAKRISHNAN, S. S. RAVI, D. J. ROSENKRANTZ, AND R. E. STEARNS, *NC-approximation schemes for NP- and PSPACE-hard problems for geometric graphs*, J. Algorithms, 26 (1998), pp. 238–274.
- [23] D. B. JOHNSON AND D. A. MALTZ, *Dynamic source routing in ad hoc wireless networks*, in Mobile Computing, T. Imielinski and H. F. Korth, eds., Kluwer Internat. Ser. Engrg. Comput. Sci. 353, Kluwer Academic Publishers, Dordrecht, The Netherlands, 1996.
- [24] P. KLEIN, *Preprocessing an undirected planar network to enable fast approximate distance queries*, in Proceedings of the 13th ACM-SIAM Symposium on Discrete Algorithms, SIAM, Philadelphia, 2002, pp. 820–827.
- [25] C. LEVCOPOULOS, G. NARASIMHAN, AND M. H. M. SMID, *Improved algorithms for constructing fault-tolerant spanners*, Algorithmica, 32 (2002), pp. 144–156.

- [26] X.-Y. LI, G. CALINESCU, AND P.-J. WAN, *Distributed construction of a planar spanner and routing for ad hoc wireless networks*, in IEEE INFOCOM 2002, New York, June 2002.
- [27] C. A. MEAD AND L. CONWAY, *Introduction to VLSI Systems*, Addison-Wesley, Reading, MA, 1980.
- [28] G. L. MILLER, S.-H. TENG, AND S. A. VAVASIS, *An unified geometric approach to graph separators*, in Proceedings of the 32nd Annual IEEE Symposium on Foundations of Computer Science, 1991, pp. 538–547.
- [29] G. NARASIMHAN AND M. SMID, *Approximating the stretch factor of Euclidean graphs*, SIAM J. Comput., 30 (2000), pp. 978–989.
- [30] C. PERKINS AND P. BHAGWAT, *Highly dynamic destination-sequenced distance-vector routing (DSDV) for mobile computers*, in Proceedings of the ACM SIGCOMM'94 Conference on Communications Architectures, Protocols and Applications, 1994, pp. 234–244.
- [31] C. E. PERKINS AND E. M. ROYER, *Ad hoc on-demand distance vector routing*, in Proceedings of the IEEE Workshop on Mobile Computing Systems and Applications, 1999, pp. 90–100.
- [32] W. D. SMITH AND N. C. WORMALD, *Geometric separator theorems and applications*, in Proceedings of the 39th IEEE Symposium on Foundations of Computer Science, 1998, pp. 232–243.
- [33] K. J. SUPOWIT, *The relative neighborhood graph with an application to minimum spanning trees*, J. ACM, 30 (1983), pp. 428–448.
- [34] J. TENENBAUM, V. DE SILVA, AND J. LANGFORD, *A global geometric framework for nonlinear dimensionality reduction*, Science, 290 (2000), p. 22.
- [35] M. THORUP, *Compact oracles for reachability and approximate distances in planar digraphs*, in Proceedings of the 42nd IEEE Symposium on Foundations of Computer Science, 2001, pp. 242–251.
- [36] M. THORUP AND U. ZWICK, *Approximate distance oracles*, in Proceedings of the ACM Symposium on Theory of Computing, 2001, pp. 183–192.
- [37] G. TOUSSAINT, *The relative neighborhood graph of a finite planar set*, Pattern Recognition, 12 (1980), pp. 261–268.
- [38] U. ZWICK, *Exact and approximate distances in graphs—a survey*, in Proceedings of the 9th Annual European Symposium on Algorithms, 2001, pp. 33–48.

A SUBEXPONENTIAL-TIME QUANTUM ALGORITHM FOR THE DIHEDRAL HIDDEN SUBGROUP PROBLEM*

GREG KUPERBERG[†]

Abstract. We present a quantum algorithm for the dihedral hidden subgroup problem (DHSP) with time and query complexity $2^{O(\sqrt{\log N})}$. In this problem an oracle computes a function f on the dihedral group D_N which is invariant under a hidden reflection in D_N . By contrast, the classical query complexity of DHSP is $O(\sqrt{N})$. The algorithm also applies to the hidden shift problem for an arbitrary finitely generated abelian group.

The algorithm begins as usual with a quantum character transform, which in the case of D_N is essentially the abelian quantum Fourier transform. This yields the name of a group representation of D_N , which is not by itself useful, and a state in the representation, which is a valuable but indecipherable qubit. The algorithm proceeds by repeatedly pairing two unfavorable qubits to make a new qubit in a more favorable representation of D_N . Once the algorithm obtains certain target representations, direct measurements reveal the hidden subgroup.

Key words. quantum algorithm, dihedral hidden subgroup

AMS subject classifications. 81P, 68W

DOI. 10.1137/S0097539703436345

1. Introduction. The hidden subgroup problem (HSP) in quantum computation takes as input a group G , a finite set S , and a black-box function (or oracle) $f : G \rightarrow S$. By promise there is a subgroup $H \subseteq G$ such that $f(a) = f(b)$ if and only if a and b are in the same (right) coset of H . The problem is to identify the subgroup H . We assume that G is given explicitly; black-box groups are a separate topic [13].

Shor's algorithm [22] solves HSP when $G = \mathbb{Z}$ in polynomial time in the length of the output. An important predecessor is Simon's algorithm [23] for the case $G = (\mathbb{Z}/2)^n$. Shor's algorithm extends to the general abelian case [14], to the case when H is normal [10], and to the case when H has few conjugates [9]. Since the main step in the generalized algorithm is the quantum character transform on the group algebra $\mathbb{C}[G]$, we will call it the *character algorithm*.

In the dihedral hidden subgroup problem (DHSP), G is the dihedral group D_N and H is generated by a reflection. (Other subgroups of D_N are only easier to find; see Proposition 2.1.) In this case H has many conjugates and the character algorithm works poorly. This hidden subgroup problem was first considered by Ettinger and Høyer [7]. They presented an algorithm that finds H with a linear number of queries (in the length of the output) but an exponential amount of computation. Ettinger, Høyer, and Knill generalized this result to the general finite hidden subgroup problem [8].

In this paper we will describe a new quantum algorithm for the dihedral group D_N with a favorable compromise between query complexity and computation time per query.

*Received by the editors October 21, 2003; accepted for publication (in revised form) January 5, 2005; published electronically October 3, 2005. This research was supported by NSF grant DMS 0072342.

<http://www.siam.org/journals/sicomp/35-1/43634.html>

[†]Department of Mathematics, University of California, Davis, CA 95616 (greg@math.ucdavis.edu).

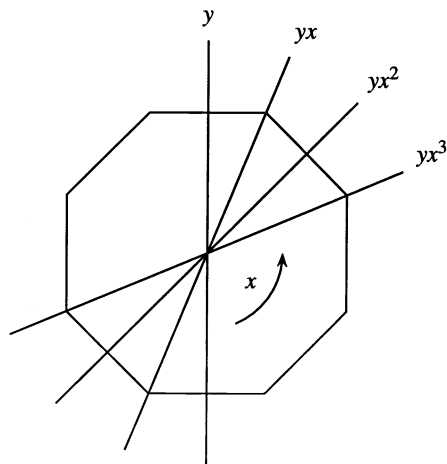


FIG. 1. Some elements of D_8 .

THEOREM 1.1. *There is a quantum algorithm that finds a hidden reflection in the dihedral group $G = D_N$ (of order $2N$) with time and query complexity $2^{O(\sqrt{\log N})}$.*

The time complexity $2^{O(\sqrt{\log N})}$ is not polynomial, but it is subexponential. By contrast any classical algorithm requires at least $2N^{1/2}$ queries on average. Unfortunately, our algorithm also requires $2^{O(\sqrt{\log N})}$ quantum space.

We will prove Theorem 1.1 in a convenient case, $N = 2^n$, in section 3. In section 5, we will provide another algorithm that works for all N , and we will obtain the sharper time and query complexity bound $\tilde{O}(3\sqrt{2^{\log_3 N}})$ when $N = r^n$ for some fixed radix r . The algorithm for this last case generalizes to many other smooth values of N .

2. Group conventions. The dihedral group D_N with $2N$ elements has the conventional presentation

$$D_N = \langle x, y \mid x^N = y^2 = yxyx = 1 \rangle.$$

(See Artin [2, section 5.3].) An element of the form x^s is a *rotation* and an element of the form yx^s is a *reflection*. The parameter s is the *slope* of the reflection yx^s . This terminology is motivated by realizing D_N as the symmetry group of a regular N -gon in the plane (Figure 1). In this model yx^s is a reflection through a line which makes an angle of $\frac{\pi s}{N}$ with the reflection line of y .

In this paper we will describe algorithms for the hidden subgroup problem with $G = D_N$ and $H = \langle yx^s \rangle$. If we know that the hidden subgroup is a reflection, then the hidden subgroup problem amounts to finding its slope s .

PROPOSITION 2.1. *Finding an arbitrary hidden subgroup H of D_N reduces to finding the slope of a hidden reflection.*

Proof. If H is not a reflection, then either it is the trivial group or it has a non-trivial intersection with the cyclic subgroup $C_N = \langle x \rangle$. Finding the hidden subgroup $H' = H \cap C_N$ in C_N is easy if we know the factors of N , and we can factor N using Shor’s algorithm. Then the quotient group H/H' is either trivial or a reflection in the quotient group G/H' .

If H is trivial, then this will be revealed by the fact that an algorithm to find the slope of a hidden reflection must fail. \square

3. A basic algorithm. In this section we will describe an algorithm to find the slope s of a hidden reflection in D_N when the period $N = 2^n$ is a power of 2. The main part of the algorithm actually finds only the parity of s . Once this parity is known, the main part can be repeated with a subgroup of D_N isomorphic to $D_{N/2}$. The group D_N has two such subgroups:

$$F_0 = \langle x^2, y \rangle, \quad F_1 = \langle x^2, yx \rangle.$$

The subgroup $F_{s \bmod 2}$ contains H and the other does not, so we can pass to one of these subgroups if and only if we know $s \bmod 2$.

For any finite set S , the notation $\mathbb{C}[S]$ denotes a Hilbert space with S as an orthogonal basis. (This is the quantum analogue of a classical data type that takes values in S .) Define the *constant pure state* $|S\rangle$ in $\mathbb{C}[S]$, or more generally in $\mathbb{C}[T]$ for any $T \supseteq S$, as the superposition

$$|S\rangle = \frac{1}{\sqrt{|S|}} \sum_{s \in S} |s\rangle.$$

For the moment let us assume an arbitrary finite hidden subgroup problem $f : G \rightarrow S$ with hidden subgroup H . Assuming that there is a classical circuit to compute f , we can dilate it to a unitary embedding

$$U_f : \mathbb{C}[G] \rightarrow \mathbb{C}[G] \otimes \mathbb{C}[S] = \mathbb{C}[G \times S]$$

which evaluates f in the standard basis:

$$U_f |g\rangle = |g, f(g)\rangle.$$

All finite hidden subgroup algorithms, including ours, begin by computing

$$U_f |G\rangle$$

and then discarding the output register $\mathbb{C}[S]$, leaving the input register for further computation. The result is the mixed state

$$\rho_{G/H} = \frac{1}{|G|} \sum |Ha\rangle \langle Ha|$$

on the input register $\mathbb{C}[G]$.

Many works on hidden subgroup algorithms describe these steps differently [22, 18, 7, 8, 9, 10]. Instead of defining U_f as an embedding that creates $f(g)$, they define it as a unitary operator that adds $f(g)$ to an ancilla. They describe its output as measured rather than discarded, and they describe the mixed state $\rho_{G/H}$ as a randomly chosen coset state $|Ha\rangle$. We have presented an equivalent description in the formalism of mixed states and quantum operations [18, Chapter 8].

Now let $G = D_N$ with $N = 2^n$. The general element of D_N is $g = y^t x^s$ with $s \in \mathbb{Z}/N$ and $t \in \mathbb{Z}/2$. Thus the input register $\mathbb{C}[D_N]$ consists of n qubits to describe s and 1 qubit to describe t . The second step of our algorithm is to apply a unitary operator to $\rho_{D_N/H}$ which is almost the character transform (section 8.2). Explicitly, we apply the quantum Fourier transform (QFT) to $|s\rangle$,

$$F_N : |s\rangle \mapsto \frac{1}{\sqrt{N}} \sum_k e^{2\pi i ks/N} |k\rangle,$$

and then measure $k \in \mathbb{Z}/N$. The measured value is uniformly random, while the state on the remaining qubit is

$$|\psi_k\rangle \propto |0\rangle + e^{2\pi i k s/N} |1\rangle.$$

(The symbol \propto means “proportional to,” so that we can omit normalization and global phase.) We will always create the same state $\rho_{D_N/H}$ and perform the same measurement, so we can suppose that we have a supply of $2^{O(\sqrt{n})}$ states $|\psi_k\rangle$, each with its own known but random value of k .

Note that $|\psi_{-k}\rangle$ and $|\psi_k\rangle$ carry equivalent information about s , because

$$(1) \quad |\psi_{-k}\rangle = X|\psi_k\rangle,$$

where X is the bit flip operator. They will be equivalent in our algorithms as well.

We would like to create the state

$$|\psi_{2^{n-1}}\rangle \propto |0\rangle + (-1)^s |1\rangle$$

because its measurement in the $|\pm\rangle$ basis reveals the parity of s . To this end we create a sieve which creates new $|\psi_k\rangle$'s from pairs of old ones. The sieve increases the number of trailing zeroes $\alpha(k)$ in the binary expansion of k . Given $|\psi_k\rangle$ and $|\psi_\ell\rangle$, their joint state is

$$|\psi_k\rangle \otimes |\psi_\ell\rangle \propto |0,0\rangle + e^{2\pi i k s/N} |1,0\rangle + e^{2\pi i \ell s/N} |0,1\rangle + e^{2\pi i (k+\ell)s/N} |1,1\rangle.$$

We now apply a CNOT gate

$$|a, b\rangle \mapsto |a, a + b\rangle$$

and measure the right qubit. The left qubit has the residual state

$$|\psi_{k\pm\ell}\rangle \propto |0\rangle + e^{2\pi i (k\pm\ell)s/N} |1\rangle$$

and the label $k \pm \ell$, which is inferred from the measurement of $a + b$. Thus we have a procedure to extract a new qubit $|\psi_{k\pm\ell}\rangle$ from the old qubits $|\psi_k\rangle$ and $|\psi_\ell\rangle$. The extraction makes an unbiased random choice between $k + \ell$ and $k - \ell$. We may well like the extracted qubit better than either of the old ones.

By iterating qubit extraction, we can eventually create the state that we like best, $|\psi_{2^{n-1}}\rangle$. We will construct a sieve that begins with $2^{\Theta(\sqrt{n})}$ qubits. Each stage of the sieve will repeatedly find two qubits $|\psi_k\rangle$ and $|\psi_\ell\rangle$ such that k and ℓ agree in $\Theta(\sqrt{n})$ low bits in addition to their trailing zeroes. With probability $\frac{1}{2}$, the label $k \pm \ell$ of the extracted qubit has \sqrt{n} more trailing zeroes than k or ℓ . If the sieve has depth $\Theta(\sqrt{n})$, we can expect it to produce copies of $|\psi_{2^{n-1}}\rangle$.

In conclusion, here is a complete description of the algorithm to find a hidden reflection in D_N with $N = 2^n$. Also let $m = \lceil \sqrt{n-1} \rceil$.

ALGORITHM 3.1. *Input:* An oracle $f : D_N \rightarrow S$ with a hidden subgroup $H = \langle yx^s \rangle$ and $N = 2^n$.

1. Make a list L_0 of copies of the state $\rho_{D_N/H}$ by applying the dilation D_f to the constant pure state $|D_N\rangle$ and discarding the input. Extract $|\psi_k\rangle$ from each $\rho_{D_N/H}$ with a QFT-based measurement.
2. For each $0 \leq j < m$, we assume a list L_j of qubit states $|\psi_k\rangle$ such that k has at least mj trailing zeroes. Divide L_j into pairs of qubits $|\psi_k\rangle$ and $|\psi_\ell\rangle$ that share at least m low bits (in addition to trailing zeroes), or $n - 1 - mj$ bits if $m = j - 1$. Extract the state $|\psi_{k\pm\ell}\rangle$ from each pair. Let the new list L_{j+1} consist of those qubit states of the form $|\psi_{k-\ell}\rangle$.

3. The final list L_m consists of states $|\psi_0\rangle$ and $|\psi_{2^{n-1}}\rangle$. Measure a state $|\psi_{2^{n-1}}\rangle$ in the $|\pm\rangle$ basis to determine the parity of the slope s .
4. Repeat steps 1–3 with the subgroup of D_N which is isomorphic to $D_{N/2}$ and which contains H .

3.1. Proof of the complexity.

THEOREM 3.2. Algorithm 3.1 requires $O(8^{\sqrt{n}})$ queries and $\tilde{O}(8^{\sqrt{n}})$ computation time.

Proof. In outline, if $|L_j| \gg 2^m$, then we can pair almost all the elements of L_j so that k and ℓ share m low bits for each pair $|\psi_k\rangle$ and $|\psi_\ell\rangle$. Then about half the pairs will form L_{j+1} , so that

$$\frac{|L_{j+1}|}{|L_j|} \approx \frac{1}{4}.$$

We can set $|L_m| = \Theta(2^m)$. Working backward, we can set $|L_0| = \Theta(8^m)$. The computation time consists of tasks with only logarithmic overhead.

In detail, we will assume that

$$|L_j| \geq C_{m-j} 2^{3m-2j}$$

for a certain constant $9 > C_k \geq 3$. We will bound the probability that this assumption survives as j increases. The constants are defined by letting $C_0 = 3$ and letting

$$C_k = \frac{C_{k-1}}{1 - 2^{-k-\frac{m}{3}}} + 2^{-2k}$$

by induction on k . It is not hard to check that

$$C_k > C_{k-1}, \quad \lim_{k \rightarrow \infty} C_k < 9.$$

(A calculator may help for the first few terms of the limit, the worst case being $m = 1$.)

Since we create L_0 directly from oracle calls, we can set

$$|L_0| = C_0 2^{3m}.$$

Given L_j , let P_j be a maximal set of pairs $|\psi_k\rangle$ and $|\psi_\ell\rangle$ with m low matching bits. Then

$$|P_j| \geq \frac{|L_j| - 2^m}{2} \geq \frac{2^{3m-2j} C_j (1 - 2^{2j-2m})}{2},$$

because there are at most 2^m unmatched pairs. The list L_{j+1} is then formed from P_j by summand extraction, so $|L_{j+1}|$ can be understood as the sum of N independent, unbiased Bernoulli random variables. In general, if B_N is a sum of N unbiased Bernoulli random variables, then

$$P[B_N \leq \frac{(1-b)N}{2}] \leq (\cosh b)^N e^{-Nb^2} \leq e^{-Nb^2/2}.$$

(The first inequality is the Chernoff bound on large deviations.) Setting

$$b = 2^{j-\frac{4m}{3}},$$

we learn that

$$|L_{j+1}| \geq \frac{2^{3m-2j}(C_j - 2^{2j-2m})(1 - 2^{j-\frac{4m}{3}})}{4} = C_{j+1}2^{3m-2j-2}$$

with probability at least

$$1 - e^{-2^{\frac{m}{3}-1}}.$$

Finally by induction on j ,

$$P[|P_j| \geq C_{m-j}2^{3m-2j} \forall j] \geq (1 - e^{-2^{\frac{m}{3}-1}})^m \rightarrow 1$$

as $m \rightarrow \infty$.

Thus the final list L_m is very likely to be large. Since the highest bit of k in $|\psi_k\rangle$ was never used for any decisions in the algorithm, it is unbiased Bernoulli for each entry of L_m . Therefore L_m is very likely to contain copies of $|\psi_{2^{n-1}}\rangle$. \square

4. Some motivation. Algorithm 3.1 can be motivated by related ideas in representation theory and the theory of classical algorithms.

On the representation theory side, the input space $\mathbb{C}[D_N]$ has an orthogonal decomposition into two-dimensional representations V_k of D_N ,

$$(2) \quad \mathbb{C}[D_N] \cong \bigoplus_{k \in \mathbb{Z}/N} V_k.$$

This means that each element of D_N is represented by a unitary operator on $\mathbb{C}[D_N]$ (given by left multiplication) and each V_k is an invariant subspace, so that each element of D_N is also represented by a unitary operator on each V_k [2, section 9.2]. Every orthogonal decomposition of a Hilbert space corresponds to a projective measurement [18, section 2.2.5]; this particular measurement can be computed using a QFT.

In the representation V_k , the generators x and y are represented as follows:

$$x \mapsto \begin{pmatrix} e^{2\pi/N} & 0 \\ 0 & e^{-2\pi/N} \end{pmatrix}, \quad y \mapsto \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}.$$

Since the state $|Ha\rangle$ is invariant under the represented action of H , the residual state $|\psi_k\rangle$ is too. Thus abstract representation theory motivates the use of this state to find H . Note also that $V_k \cong V_{-k}$ as representations, as if reflected in the equivalence between $|\psi_k\rangle$ and $|\psi_{-k}\rangle$ in (1).

The representation V_k is irreducible except when $k = 0$ or $k = N/2$. Thus (2) is not far from the Burnside decomposition of $\mathbb{C}[G]$ into irreducible representations in the special case $G = D_N$. When expressed as a unitary operator, the Burnside decomposition is called the *character transform* or the noncommutative Fourier transform. (Measuring the character name solves the hidden subgroup problem for normal subgroups [10] and almost normal subgroups [9].) Use of $V_{N/2}$ as the target of Algorithm 3.1 is motivated by its reducibility; the measurement corresponding to its irreducible decomposition is the one that reveals the slope of s .

On the algorithm side, the sieve in Algorithm 3.1 is similar to a sieve algorithm for a learning problem due to Blum, Kalai, and Wasserman [5] and to a sieve to find shortest vector in a lattice due to Ajtai, Kumar, and Sivakumar [1].

Ettinger and Høyer [7] observed that if the state $|\psi_k\rangle$ for the hidden subgroup $H = \langle x^s y \rangle$ will be found in the state $|\psi'_k\rangle$ for a reference subgroup $H' = \langle x^t y \rangle$ with probability

$$\cos(\pi i(s-t)k/N)^2.$$

Thus the state $|\psi_k\rangle$ can provide a coin flip with this bias. We call such a coin flip a *cosine observation* of the slope s . Ettinger and Høyer showed that s is revealed by a maximum likelihood test with respect to $O(\log N)$ cosine observations with random values of k . They suggested a brute-force search to solve this maximum likelihood problem. Our first version of Algorithm 3.1 was a slightly subexponential, classical sieve on cosine observations that even more closely resembles the Blum–Kalai–Wasserman algorithm. Replacing the cosine observations by the qubit states $|\psi_k\rangle$ themselves significantly accelerates the algorithm.

5. Other algorithms. Algorithm 3.1 presents a simplified sieve which is close to the author’s original thinking. But it is neither optimal nor fully general. In this section we present several variations which are faster or more general.

The first task is to prove Theorem 1.1 when N is not a power of 2. Given any qubit state $|\psi_k\rangle$, we can assume that $0 \leq k \leq \frac{N}{2}$, since $|\psi_k\rangle$ and $|\psi_{-k}\rangle$ are equivalent. The list L_j will consist of qubits $|\psi_k\rangle$ with

$$0 \leq k < 2^{m^2-mj+1},$$

where

$$m = \lceil \sqrt{(\log_2 N) - 2} \rceil.$$

Another difference when N is not a power of 2 is that the quantum Fourier transform on \mathbb{Z}/N is more complicated. An efficient approximate algorithm was given by Kitaev [14]; another algorithm which is exact (in a sense) is due to Mosca and Zalka [17].

ALGORITHM 5.1. *Input: An oracle $f : D_N \rightarrow S$ with a hidden subgroup $H = \langle yx^s \rangle$.*

1. *Make a list L_0 of copies of $\rho_{D_N/H}$. Extract a qubit state $|\psi_k\rangle$ from each $\rho_{D_N/H}$ using a QFT on \mathbb{Z}/N and a measurement.*
2. *For each $0 \leq j < m$, we assume a list L_j of qubit states $|\psi_k\rangle$ such that $0 \leq k \leq 2^{m^2-mj+1}$. Randomly divide L_j into pairs of qubits $|\psi_k\rangle$ and $|\psi_\ell\rangle$ that such that*

$$|k - \ell| \leq 2^{m^2-m(j+1)+1}.$$

Let the new list L_{j+1} consist of those qubit states of the form $|\psi_{|k-\ell|}\rangle$.

3. *The final list L_m consists of states $|\psi_0\rangle$ and $|\psi_1\rangle$. Perform the Ettinger–Høyer measurement on the copies of $|\psi_1\rangle$ with different values of t to learn $s \in \mathbb{Z}/N$ to within $N/4$.*
4. *Write $N = 2^a M$ with M odd. By the Chinese remainder theorem,*

$$C_N \cong C_{2^a} \times C_M.$$

For each $1 \leq j \leq \lceil \log_2 N \rceil$, apply Algorithm 3.1 to produce many $|\psi_k\rangle$ with $2^{\min(a,j)}|k|$. Then repeat steps 1–4 after applying the group automorphism $x \mapsto x^{2^{-j}}$ to the C_M factor of D_N . This produces copies of $|\psi_{2^j}\rangle$, and hence cosine observations $\cos(\pi i 2^j(s-t)/N)^2$. These observations determine s .

TABLE 1
Average canceled bits in a simulation (100 trials).

Queries	3	3 ²	3 ³	3 ⁴	3 ⁵	3 ⁶	3 ⁷	3 ⁸
Zeroed bits	3.62	6.75	12.53	19.07	27.14	36.44	47.51	59.76
$\sqrt{2 \log_3 2^n}$	2.14	2.92	3.98	4.91	5.85	6.78	7.74	8.68

The proof of Theorem 3.2 carries-over to show that Algorithm 5.1 also requires only $O(8\sqrt{\log_2 N})$ queries and quasi-linear time in its data. The only new step is to check that in the final list L_m , the qubit states $|\psi_0\rangle$ and $|\psi_1\rangle$ are almost equally likely. This is a bit tricky but inevitable, given that the lowest bit of k can be almost uncorrelated with the way that $|\psi_k\rangle$ is paired.

Remark 5.2. Høyer described a simplification of Algorithm 5.1 [11]. Given only one copy each of

$$|\psi_1\rangle, |\psi_2\rangle, \dots, |\psi_{2^k}\rangle,$$

with $2^k \geq N$, the slope s can be recovered directly by a quantum Fourier transform. More precisely, the measured Fourier number t of these qubits reveals s by the relation

$$\frac{t}{2^k} \sim \frac{s}{N}.$$

This simplification saves a factor of $O(\log N)$ computation time.

Now suppose that $N = r^n$ for some small radix r ; Algorithm 3.1 generalizes to this case with only slight changes. It is natural to accelerate it by recasting it as a greedy algorithm. To this end, we define an objective function $\alpha(k)$ that expresses how much we like a given state $|\psi_k\rangle$. Namely, let αk be the number of factors of r in k with the exception that $\alpha(0) = 0$. Within the list L of qubit states available at any given time, we will greedily pick $|\psi_k\rangle$ and $|\psi_\ell\rangle$ to maximize $\alpha(k \pm \ell)$. It is also natural to restrict our greed to the qubits that minimize α , because there is no advantage to postponing their use in the sieve.

ALGORITHM 5.3. *Input: An oracle $f : D_N \rightarrow S$ with a hidden subgroup $H = \langle yx^s \rangle$ and $N = r^n$.*

1. *Make a list L of qubit states $|\psi_k\rangle$ extracted from copies of $\rho_{D_N/H}$.*
2. *Within the sublist L' of L that minimizes α , repeatedly extract $|\psi_\ell\rangle$ from a pair of qubits $|\psi_k\rangle$ and $|\psi_\ell\rangle$ that maximize $\alpha(k \pm \ell)$.*
3. *After enough qubits $|\psi_k\rangle$ appear with $\frac{N}{r}|k$, measure $s \pmod r$ using state tomography. Then repeat the algorithm with a subgroup of D_N isomorphic to $D_{N/r}$.*

The behavior of Algorithm 5.3 (but not its quantum state) can be simulated by a classical randomized algorithm. We include the source code of a simulator written in Python with this article [15] with $r = 2$. Our experiments with this simulator led to a false conjecture for algorithm’s precise query complexity. Nonetheless we present some of its results in Table 1. The last line of the table is roughly consistent with Theorem 5.4. Note that the sieve is a bit more efficient when $r = 2$ because then $k \pm \ell$ increases by 1 in the unfavorable case and at least 2 in the favorable case.

THEOREM 5.4. *Algorithm 5.3 requires $\tilde{O}(3\sqrt{2 \log_3 N})$ queries and quasi-linear time in the number of queries.*

Here is a heuristic justification of the query bound in Theorem 5.4. We assume, as the proof will, that $r = 3$ and $N = 3^n$. Then with $3\sqrt{2^n}$ queries, we can expect

qubit extraction to initially cancel about $\sqrt{2n}$ ternary digits (trits) with probability $\frac{1}{2}$. If we believe the query estimate for $n' < n$, then we can expect the new qubit to be about 3 times as valuable as the old one, since

$$\sqrt{2n} - \sqrt{2n - \sqrt{2n}} \approx 1.$$

Such a qubit extraction trades 2 qubits for 1 qubit, which is half the time equivalent to the original 2 and half the time 3 times as valuable. Thus each step of the sieve breaks even; it is like a gamble with \$2 that is equally likely to return \$1 or \$3.

Sketch of proof. We will show that the sieve produces states $|\psi_{a_{N/r}}\rangle$ (which we will call *final states*) with adequate probability when provided with at least $Cn3^{\sqrt{2\log_3 N}}$ queries. The work per query is quasi-linear in $|L|$ (initially the number of queries) if the list L is dynamically sorted. To simplify the formulas, we assume that $r = 3$, although the proof works for all r .

We can think of a qubit state $|\psi_k\rangle$ as a monetary asset, valued by the function

$$V(k) = 3^{-\sqrt{2(n-1-\alpha(k))}}.$$

Thus the total value $V(L)$ of the initial list L is at least

$$V(L) \geq Cn.$$

We claim that over a period of the sieve that increases $\min \alpha$ by 1, the expected change in $V(L)$ is at worst $-C$. Since $\min \alpha$ can only increase $n - 1$ times, $V(L) \geq C$ when $\min \alpha = n - 1$. Thus the sieve produces at least C final states on average. Along the way, the changes to $V(L)$ are independent (but not identically distributed) Bernoulli trials. One can show using a version of the Chernoff bound (as in the proof of Theorem 3.2) that the number of final states is not maldistributed. We will omit this refinement of the estimates and spell out the expected behavior of $V(L)$.

Given k , let

$$\beta = \beta(k) = n - 1 - \alpha(k)$$

for short, so that β can be thought of as the number of uncanceled trits in the label k of $|\psi_k\rangle$. Suppose that two labels k and ℓ or $-\ell$ share m trits in addition to $\alpha(k)$ cancelled trits. Then

$$(3) \quad V(k) = V(\ell) = 3^{-\sqrt{2\beta}}.$$

The state $|\psi_{k\pm\ell}\rangle$ extracted from $|\psi_k\rangle$ and $|\psi_\ell\rangle$ has the expected value

$$(4) \quad \begin{aligned} E[V(k \pm \ell)] &= \frac{3^{-\sqrt{2\beta}} + 3^{-\sqrt{2(\beta-m)}}}{2} \\ &> 2V(k) \frac{1 + 3^{m/\sqrt{2\beta}}}{4}, \end{aligned}$$

using the elementary relation

$$\sqrt{2\beta} - \sqrt{2(\beta-m)} = \frac{2m}{\sqrt{2\beta} + \sqrt{2(\beta-m)}} > \frac{m}{\sqrt{2\beta}}.$$

The most important feature of (4) is that if $m > \sqrt{2\beta}$, the expected change in $V(L)$ is positive. Thus in bounding the attrition of $V(L)$, we can assume that $m \leq \sqrt{2\beta}$ for the best-matching qubits $|\psi_k\rangle$ and $|\psi_\ell\rangle$ in the sublist L' that minimizes α . By the pigeonhole principle, this can happen only when

$$|L'| \leq 3\sqrt{2\beta}.$$

(To apply the pigeonhole principle properly, use the equivalence between $|\psi_k\rangle$ and $|\psi_{-k}\rangle$ to assume that the first nonzero digit is 1. There are then 3^m choices for the next m digits.)

When qubit extraction decreases $V(L)$, it decreases by at worst the value of one parent, given by the right side of (3). Likewise, if $|L'| = 1$ and its unique element $|\psi_k\rangle$ must be discarded, the loss to $V(L)$ is again the right side of (3). Thus the total expected loss as L' is exhausted is at most

$$3^{-\sqrt{2\beta}} 3^{\sqrt{2\beta}} < 1.$$

We can therefore take $C = 1$, although a larger C may be convenient to facilitate the Chernoff bound. \square

Remark 5.5. A close examination of Algorithm 5.3 and Theorem 5.4 reveals that the sieve works with the same complexity bound if N factors as

$$N = N_1 N_2 \dots N_m$$

and N_k is within a bounded factor of 3^k . In this case the sieve will determine $s \pmod{N_1}$. This is enough values of N to extend to an algorithm for all N by the method of spliced approximation section 7.

6. Generalized dihedral groups and hidden shifts. In this section we consider several other problems that are equivalent or closely related to the hidden dihedral subgroup problem.

In general if A is an abelian group, let $\exp(A)$ denote the multiplicative form of the same group. Let $C_n = \exp(\mathbb{Z}/n)$ be the multiplicative cyclic group of order n . If A is any abelian group, define the *generalized dihedral group* to be the semidirect product

$$D_A \cong C_2 \ltimes \exp(A)$$

with the conjugation relation

$$x^{-1} = yxy$$

for all $x \in \exp(A)$ and for the nontrivial $y \in C_2$. Any element of the form yx is a *reflection* in D_A .

Suppose that A is an abelian group and $f, g : A \rightarrow S$ are two injective functions that differ by a shift:

$$f(a) = g(a + s).$$

Then the task of finding s from f and g is the *abelian hidden shift problem*. Another problem is the hidden reflection problem in A (as opposed to in D_A). In this problem, $f : A \rightarrow S$ is a function which is injective except that

$$f(a) = f(s - a)$$

TABLE 2
An oracle that hides $\langle yx^3 \rangle$ in D_8 and its hidden shift.

a	1	x	x^2	x^3	x^4	x^5	x^6	x^7
$f(a)$	A	B	C	D	E	F	G	H
a	y	yx	yx^2	yx^3	yx^4	yx^5	yx^6	yx^7
$f(a)$	F	G	H	A	B	C	D	E

for some hidden s .

PROPOSITION 6.1. *If A is an abelian group, the hidden shift and hidden reflection problems in A are equivalent to the hidden reflection problem in D_A .*

See Table 2 for an example.

Proof. If $a \in A$, let x^a denote the corresponding element in $\exp(A)$. Given $f, g : A \rightarrow S$, define

$$h(x^a) = f(a), \quad h(yx^a) = g(a).$$

Then evidently

$$h(x^a) = h(yx^{s+a})$$

if and only if

$$f(a) = g(a + s).$$

We can also reduce the pair f and g to a function with a hidden reflection. Namely, let $S^{(2)}$ be the set of unordered pairs of elements of S and define $h : A \rightarrow S^{(2)}$ by

$$h(a) = \{f(-a), g(a)\}.$$

Then h is injective save for the relation

$$h(a) = h(s - a).$$

Conversely, suppose that $h : A \rightarrow S$ is injective save for the relation

$$h(a) = h(s - a).$$

If there is a $v \in A$ such that $2v \neq 0$, define

$$f : A \rightarrow S^{\times 2}, \quad g : A \rightarrow S^{\times 2}$$

by

$$f(a) = (h(-a), h(v - a)), \quad g(a) = (h(a), h(a - v)).$$

(If A is cyclic, we can just take $v = 1$.) Then f and g are injective and

$$f(a) = g(a + s).$$

If all $v \in A$ satisfy $2v = 0$, then h hides a subgroup of A generated by s , so we can find s by Simon's algorithm. \square

Note also that Proposition 2.1 generalizes readily to generalized dihedral subgroups: finding a hidden reflection in D_A is as difficult as finding any hidden subgroup.

A final variation of DHSP is the hidden substring problem. In the $N \hookrightarrow M$ hidden substring problem,

$$\begin{aligned} f &: \{0, 1, 2, \dots, N - 1\} \rightarrow S, \\ g &: \{0, 1, 2, \dots, M - 1\} \rightarrow S \end{aligned}$$

are two injective functions such that f is a shifted restriction of g , i.e.,

$$f(x) = g(x + s)$$

for all $0 \leq x < N$ and for some fixed $0 \leq s < M - N$.

7. More algorithms. In this section we will establish a generalization of Theorem 1.1 and a corollary.

THEOREM 7.1. *The abelian hidden shift problem has an algorithm with time and query complexity $2^{O(\sqrt{n})}$, where n is the length of the output, uniformly for all finitely generated abelian groups.*

COROLLARY 7.2. *The $N \hookrightarrow 2N$ hidden substring problem has an algorithm with time and query complexity $2^{O(\sqrt{\log N})}$.*

The proof of Corollary 7.2 serves as a warm-up to the proof of Theorem 7.1. It introduces a technique for converting hidden shift algorithms that we call *spliced approximation*.

Proof of Corollary 7.2. Identify the domain of f with \mathbb{Z}/N (no matter that this identification is artificial). Make a random estimate t for the value of s , and define $h : D_N \rightarrow S$ by

$$g'(n) = g(n + t).$$

If t is a good estimate for s , then f and g' approximately hide the hidden shift $s - t$. If we convert f and g to a function $h : D_N \rightarrow S$, then apply its dilation U_h with input $|D_N\rangle$ and discard the output, the result is a state $\rho_h = \rho_{f,g'}$ which is close to the state $\rho_{D_N/H}$ used in Algorithm 5.1.

We need to quantify how close. The relevant metric on states for us is the trace distance [18, section 9.2]. In general if ρ and ρ' are two states on a Hilbert space \mathcal{H} , the trace distance $\|\rho - \rho'\|$ is the maximum probability that any measurement, indeed any use in a quantum algorithm, will distinguish them. In our case,

$$\|\rho_h - \rho_{D_N/H}\| = \frac{|s - t|}{N}.$$

If

$$\frac{|s - t|}{N} = 2^{-O(\sqrt{\log N})},$$

then with bounded probability, Algorithm 5.1 will never see the difference between ρ_h and $\rho_{D_N/H}$. Thus $2^{O(\sqrt{\log N})}$ guesses for s suffice. \square

A second warm-up to the general case of Theorem 7.1 is the special case $A = \mathbb{Z}$. Recall that more computation is allowed for longer output. Suppose that the output

has n bits, i.e., the shift s is at most 2^n . In the language of deterministic hiding, we restrict the domain of $f, g : \mathbb{Z} \rightarrow S$ to the set $\{0, 1, 2, \dots, 2^m\}$, where $m = n + \Theta(\sqrt{n})$, and interpret this set as $\mathbb{Z}/2^m$. Then f and g approximately differ by the shift s . If we form the state $\rho_{f,g}$ as in the proof of Corollary 7.2, then its trace distance from the state $\rho_{D_N/H}$, with $N = 2^m$, is $2^{-O(\sqrt{n})}$. Thus Algorithm 5.1 will never see the states differ.

Sketch of proof of Theorem 7.1. In the general case, the classification of finitely generated abelian groups says that

$$A \cong \mathbb{Z}^b \oplus \mathbb{Z}/N_1 \oplus \mathbb{Z}/N_2 \oplus \dots \oplus \mathbb{Z}/N_a.$$

Assuming a bound on the length of the output, we can truncate each \mathbb{Z} summand of A , as in the case $A = \mathbb{Z}$. (We suppose that we know how many bits of output are allocated to each free summand of A .) Thus we can assume that

$$A = \mathbb{Z}/N_1 \oplus \mathbb{Z}/N_2 \oplus \dots \oplus \mathbb{Z}/N_a,$$

and the problem is to find s in time $2^{O(\sqrt{\log |A|})}$. In other words, the problem is to solve HSP for a finite group D_A .

The general element of D_A can be written $y^t x^a$ with $t \in \mathbb{Z}/2$ and $a \in A$. Following the usual first step, we can first prepare the state $\rho_{D_A/H}$. Then we can perform a quantum Fourier transform on each factor of A , then measure the answer, to obtain a label

$$k = (k_1, k_2, \dots, k_a) \in A$$

and a qubit state

$$|\psi_k\rangle \propto |0\rangle + e^{2\pi i \sum_j s_j k_j / N_j} |1\rangle.$$

(As in section 4, this state is H -invariant in a two-dimensional representation V_k of D_A .) We will outline a sieve algorithm to compute any one coordinate of the slope, without loss of generality s_a .

As in Algorithm 5.3, we will guide the behavior of the sieve by an objective function α on A . Given k , let $b(k)$ be the first j such that $k_j \neq 0$. If $b < a$, then let

$$\alpha(k) = \sum_{j=1}^b [1 + \log_2(N_j + 1)] - \lceil \log_2(k_b + 1) \rceil.$$

If $b = a$, then let

$$\alpha(k) = \sum_{j=1}^a [1 + \log_2(N_j + 1)].$$

As in Algorithm 5.3, we produce a list L of $2^{O(\sqrt{\log |A|})}$ qubits with states $|\psi_k\rangle$. Within the minimum of α on L , we repeatedly find pairs $|\psi_k\rangle$ and $|\psi_\ell\rangle$ that maximize $\alpha(k + \ell)$ or $\alpha(k - \ell)$, then we extract $|\psi_{k+\ell}\rangle$ from each such pair. The end result is a list of qubit states $|\psi_k\rangle$ with

$$k = (0, 0, \dots, 0, k_a).$$

The set of k of this form is closed under sums and differences, so we can switch to Algorithm 5.1 to eventually determine the slope s_a . \square

Note that many abelian groups A are not very different from cyclic groups, so that the generalized dihedral group D_A can be approximated for our purposes by a standard dihedral group. For example, if $A \cong \mathbb{Z}^a$ is free abelian with many bits of output allocated to each coordinate, then we can pass to a truncation

$$\mathbb{Z}/N_1 \oplus \mathbb{Z}/N_2 \oplus \cdots \oplus \mathbb{Z}/N_a$$

with relatively prime N_j 's. In this case the truncation is cyclic.

8. Hidden subgroup generalities. In this section we will make some general observations about quantum algorithms for hidden subgroup problems. Our comments are related to work by Hallgren, Russell, and Ta-Shma [10] and by Grigni et al. [9].

8.1. Quantum oracles. The first step of all quantum algorithms for the hidden subgroup problem is to form the state $\rho_{G/H}$, or an approximation when G is infinite, except when the oracle $f : G \rightarrow S$ has special properties.

Suppose that a function $f : G \rightarrow S$ that hides the subgroup H . We can say that f *deterministically* hides H because it is a deterministic function. Some problems in quantum computation might reduce to a nondeterministic oracle $f : G \rightarrow \mathcal{H}$, where \mathcal{H} is a Hilbert space. We say that such an f *orthogonally* hides H if f is constant on each right coset Ha of H and orthogonal on distinct cosets. If a quantum algorithm invokes the dilation D_f of f and then discards the output, then it solves the orthogonal hidden subgroup problem as well as the deterministic one.

Computing D_f and discarding its output can also be viewed as a quantum oracle. A general quantum computation involving both unitary and nonunitary actions can be expressed as a quantum operation [18, Chapter 8]. In this case the operation is a map $\mathcal{E}_{G/H}$ on $\mathcal{M}(\mathbb{C}[G])$, where in general $\mathcal{M}(\mathcal{H})$ denotes the algebra of operators on a Hilbert space \mathcal{H} . It is defined by

$$\mathcal{E}_{G/H}(|a\rangle\langle b|) = \begin{cases} |a\rangle\langle b| & \text{if } Ha = Hb, \\ 0 & \text{if } Ha \neq Hb. \end{cases}$$

We say that the quantum oracle $\mathcal{E}_{G/H}$ *projectively* hides the subgroup H . Unlike deterministic and orthogonal oracles, the projective oracle is uniquely determined by H . Again, all quantum algorithms for hidden subgroup problems work with this more difficult oracle.

Finally, if G is finite, the projective oracle $\mathcal{E}_{G/H}$ can be applied to the constant pure state $|G\rangle$ to produce the state

$$\rho_{G/H} = \frac{|H|}{|G|} \sum |Ha\rangle\langle Ha|.$$

So an algorithm could use a no-input oracle that simply broadcasts copies of $\rho_{G/H}$. Such an oracle *coherently* hides H . This oracle has been also been called the random coset oracle [20] because the state $\rho_{G/H}$ is equivalent to the constant pure state $|Ha\rangle$ on a uniformly randomly chosen coset. Almost all existing quantum algorithms for finite hidden subgroup problems need only copies of the state $\rho_{G/H}$. Algorithms 3.1 and 5.3 are exceptions: They use $\rho_{D_N/H}$ to find the parity of the slope s and then rely on $\mathcal{E}_{D_N/H}$ with other inputs (constant pure states on subgroups) for later stages.

The possibly slower algorithm, Algorithm 5.1, works with the coherent oracle; it uses only $\rho_{D_N/H}$.

The distinctions between deterministic, orthogonal, and projective hiding apply to any hidden partition problem. In one special case, called the hidden stabilizer problem [14], a group G acts transitively on a set S and a function $f : S \rightarrow T$ is invariant under a subgroup $H \subseteq G$. The hidden stabilizer problem has enough symmetry to justify consideration of coherent hiding. It would be interesting to determine when one kind of hiding is harder than another. For example, if f is injective save for a single repeated value, then there is a sublinear algorithm for deterministic hiding [6]. But projective hiding requires at least linear time and we do not know an algorithm for coherent hiding which is faster than quadratic time.

In a variant of coherent HSP, the oracle outputs nonuniform mixtures of coset states $|Ha\rangle$. The mixtures may even be chosen adversarially. This can make the subgroup H less hidden, for example, in the trivial extreme in which the state is $|H\rangle$ with certainty. At the other extreme, we can always uniformize the state by translating by a random group element. Thus uniform coherent HSP is the hardest representative of this class of problems.

8.2. The character measurement. The second step of all quantum algorithms for the generic hidden subgroup problem is to perform the character measurement. (The measurement in our algorithms is only trivially different.) The result is the name or character of an irreducible unitary representation (or irrep) V and a state in V . Mathematically the character measurement is expressed by the Burnside decomposition of the group algebra $\mathbb{C}[G]$ as a direct sum of matrix algebras [21]:

$$\mathbb{C}[G] \cong \bigoplus_V \mathcal{M}(V).$$

Here $\mathcal{M}(V)$ is the algebra of operators on the irrep V ; the direct sum runs over one representative of each isomorphism type of unitary irreps. The group algebra $\mathbb{C}[G]$ has two commuting actions of G , given by left and right multiplication, and with respect to these two actions,

$$\mathcal{M}(V) \cong V \otimes V^*,$$

so that the Burnside decomposition can also be written

$$(5) \quad \mathbb{C}[G] \cong \bigoplus_V V \otimes V^*.$$

In light of the identification with matrices, the factor of V^* is called the *row space*, while the factor of V is the *column space*.

The Burnside decomposition is also an orthogonal decomposition of Hilbert spaces and so corresponds to a projective measurement on $\mathbb{C}[G]$. This is the character measurement. A character transform is an orthonormal change of basis that refines equation (5). Its precise structure as a unitary operator depends on choosing a basis for each V .

The state $\rho_{G/H}$ has an interesting structure with respect to the Burnside decomposition. In general if \mathcal{H} is a finite-dimensional Hilbert space, let $\rho_{\mathcal{H}}$ denote the uniform mixed state on \mathcal{H} ; if V is a representation of a group G , let V^G denote its invariant space. It is easy to check that

$$\rho_{G/H} = \rho_{\mathbb{C}[G]^H},$$

where G (and therefore H) acts on $\mathbb{C}[G]$ by left multiplication. In the Burnside decomposition, the left multiplication action on each $V \otimes V^*$ is trivial on the right factor V^* and is just the defining action of G on V . Since $\rho_{G/H}$ is the uniform state on all H -invariant vectors in $\mathbb{C}[G]$, this property descends through the Burnside decomposition:

$$\rho_{G/H} = \bigoplus_V \rho_{V^H} \otimes \rho_{V^*}.$$

This relation has two consequences. First, as has been noted previously [9], the state on the row space V^* has no useful information. Second, since $\rho_{G/H}$ decomposes as a direct sum with respect to the Burnside decomposition, the character measurement sacrifices no coherence to the environment; it only measures something that the environment already knows. Our reasoning here establishes the following proposition.

PROPOSITION 8.1. *Let G be a finite group and assume an algorithm or oracle to compute the character transform on $\mathbb{C}[G]$. Then a process provides the state $\rho_{G/H}$ is equivalent to a process that provides the name of an irrep V and the state ρ_{V^H} with probability*

$$P[V] = \frac{(\dim V)(\dim V^H)|H|}{|G|}.$$

Proposition 8.1 sharpens the motivation to work with irreps in the hidden subgroup problem. If you obtain the state $\rho_{G/H}$, and if you can efficiently perform the character measurement on states, then you might as well apply it to $\rho_{G/H}$.

Proposition 8.1 and the definition of coherent HSP in section 8.1 suggest another class of oracles related to the hidden subgroup problem. In general an oracle might provide the name of a representation V and a state ρ which is some mixture of H -invariant pure states in V . It is tempting to describe such a ρ as H -invariant, but technically that is a weaker condition that also applies to other states. For example, the uniform state on V is H -invariant. So we say that ρ is *purely H -invariant* if it is supported on the H -invariant space V^H . For example, the uniform state $\rho_{G/H}$ is purely H -invariant. More generally the purely H -invariant states on $\mathbb{C}[G]$ are exactly the mixtures of constant pure states of right cosets $|Ha\rangle$.

PROPOSITION 8.2. *Let G be a finite group. Then any purely H -invariant state ρ on $\mathbb{C}[G]$ can be converted to $\rho_{G/H}$. In the presence of an algorithm or oracle to perform the character transform on $\mathbb{C}[G]$, any purely H -invariant state ρ on any irrep V can be converted to $\rho_{G/H}$.*

Proof. If we right-multiply ρ by a uniformly random element of G , it becomes $\rho_{G/H}$. If we perform the reverse character transform to a purely H -invariant state ρ on V , it becomes a purely H -invariant state on $\rho_{G/H}$ itself. \square

The message of Proposition 8.2 is that the uniform mixture $\rho_{G/H}$ reveals the least information about H among all mixtures of coset states $|Ha\rangle$. The distribution on irreps V described in Proposition 8.1, together with the uniform state on V^H , also reveals the least information about H among all such distributions.

9. A general algorithm. In this section we will discuss a general algorithm for coherent HSP for an arbitrary finite group G and an arbitrary subgroup H . It is an interesting abstract presentation of all the algorithms for dihedral groups in this paper. Unfortunately, it might not be directly useful for any groups other than dihedral groups.

The algorithm uses the definitions and methods of section 8.2, together with a generalized notion of summand extraction. In general if V and W are two unitary representations of G , their tensor product decomposes as an orthogonal direct sum of irreps with respect to the diagonal action of G :

$$(6) \quad V \otimes W \cong \bigoplus_X \mathcal{H}_X^{W,V} \otimes X.$$

Here again the direct sum runs over one representative of each isomorphism class of irreps. The Hilbert space $\mathcal{H}_X^{W,V}$ is the *multiplicity factor* of the decomposition; its dimension is the number of times that X arises as a summand of $V \otimes W$. The decomposition defines a partial measurement of the joint Hilbert space $V \otimes W$, which extracts X (and $\mathcal{H}_X^{W,V}$). If V and W carry purely H -invariant states, then the state on X is also purely H -invariant.

ALGORITHM 9.1. *Input: An oracle that produces $\rho_{G/H}$.*

1. *Make a list L of copies of $\rho_{G/H}$. Extract an irrep V with a purely H -invariant state from each copy.*
2. *Choose an objective function α on $\text{Irrep}(G)$, the set of irreps of G .*
3. *Find a pair of irreps V and W in L such that $\alpha(V)$ and $\alpha(W)$ are both low, but such that α is significantly higher for at least one summand of $V \otimes W$. Extract an irreducible summand X from $V \otimes W$ and replace V and W in L with X . Discard the multiplicity factor.*
4. *Repeat step 3 until α is maximized on some irrep V . Perform tomography on V to reveal useful information about H .*
5. *Repeat steps 2–4 to fully identify H .*

For any given group G , Algorithm 9.1 requires subalgorithms to compute the character measurement (5) and the tensor decomposition measurement (6). Efficient algorithms for character measurements and character transforms are a topic of active research [4, 16] and are unknown for many groups. We observe that tensor decomposition measurement at least reduces to the character measurement.

PROPOSITION 9.2. *Let V and W be irreducible representations of a finite group G . If group operations in G and summand extraction from $\mathbb{C}[G]$ are both efficient, then summand extraction from $V \otimes W$ is also efficient.*

Proof. Embed V and W into separate copies of $\mathbb{C}[G]$ in a G -equivariant way. Then apply the unitary operator

$$U(|a\rangle \otimes |b\rangle) = |b^{-1}a\rangle \otimes |b\rangle$$

to $\mathbb{C}[G] \otimes \mathbb{C}[G]$. The operator U transports left multiplication by the diagonal subgroup $G_\Delta \subset G \times G$ to left multiplication by G on the right factor. Then summand extraction from the right factor of $\mathbb{C}[G] \otimes \mathbb{C}[G]$ is equivalent to summand extraction from $V \otimes W$, since, after U is applied, the group action on the right factor of $\mathbb{C}[G] \otimes \mathbb{C}[G]$ coincides with the diagonal action on $V \otimes W$. \square

In light of Beals's algorithm to compute a character transform on the symmetric group [4] and Proposition 9.2, Algorithm 9.1 may look promising when $G = S_n$ is the symmetric group. But the algorithm seems to work poorly for this group, because the typical irrep V of S_n is very large. Consequently the decomposition (6) typically involves many irreps of S_n . This offers very little control for a sieve.

Note that if Algorithm 9.1 were useful for the symmetric group, its time complexity would be $2^{O(\sqrt{\log |G|})}$ at best. This is the same complexity class as a known classical algorithm for the graph isomorphism or automorphism problem [3], which

is the original motivation for the symmetric hidden subgroup problem (SHSP). We believe that general SHSP is actually much harder than graph isomorphism. If graph isomorphism does admit a special quantum algorithm, it could be analogous to a quantum polynomial time algorithm found by van Dam, Hallgren, and Ip [24] for certain special abelian hidden shift problems. (In particular their algorithm applies to the Legendre symbol with a hidden shift.) All these problems have special oracles f that allow faster algorithms.

One reason that SHSP looks hard is that symmetric groups have many different kinds of large subgroups. For example, if p_1, p_2, \dots, p_n is a set of distinct primes, then

$$D_{p_1 p_2 \dots p_n} \hookrightarrow S_{p_1 + p_2 + \dots + p_n}$$

(exercise). Thus DHSP reduces to SHSP. Hidden shift in the symmetric group also reduces to SHSP (exercise).

The sieve of Algorithm 9.1 looks the most promising when the group G is large but $V \otimes W$ always has few terms. This is similar to demanding that most or all irreps of G are low-dimensional. So suppose that all irreps have dimension at most k and consider the limit $|G| \rightarrow \infty$ for fixed k . Isaacs and Passman [12] showed that there is a function $f(k)$ such that if all irreps have dimension at most k , then G has an abelian subgroup $\exp(A)$ of index at most $f(k)$. By the reasoning of Proposition 2.1, the hardest hidden subgroup H for a such a G is one which is disjoint from $\exp(A)$ (except for the identity). But by the reasoning of section 6, any such hidden subgroup problem reduces to the hidden shift problem on A . The generalized sieve of Algorithm 9.1 is not as fast as the dihedral sieve on D_A .

Acknowledgments. Some elements of the algorithms in this article are due to Ettinger and Høyer [7]. Regev has presented some related ideas related to lattice problems [20] and more recently has found a space-efficient variation of the algorithms in this article [19]. (We have also borrowed some aspects of his exposition of our algorithm.) The author would like to thank Robert Beals, Robert Guralnick, Peter Høyer, and Eric Rains for useful discussions. The author would also like to thank the referees for useful comments.

REFERENCES

- [1] M. AJTAI, R. KUMAR, AND D. SIVAKUMAR, *A sieve algorithm for the shortest lattice vector problem*, in Proceedings of the 33rd Annual ACM Symposium on Theory of Computing, 2001, pp. 601–610.
- [2] M. ARTIN, *Algebra*, Prentice-Hall Inc., Englewood Cliffs, NJ, 1991.
- [3] L. BABAI AND E. M. LUKS, *Canonical labeling of graphs*, in Proceedings of the 15th Annual ACM Symposium on Theory of Computing, ACM Press, New York, 1983, pp. 171–183.
- [4] R. BEALS, *Quantum computation of Fourier transforms over symmetric groups*, in ACM Symposium on Theory of Computing, 1997, pp. 48–53.
- [5] A. BLUM, A. KALAI, AND H. WASSERMAN, *Noise-tolerant learning, the parity problem, and the statistical query model*, J. ACM, 50 (2003), pp. 506–519.
- [6] H. BUHRMAN, C. DÜRR, M. HEILIGMAN, P. HØYER, F. MAGNIEZ, M. SANTHA, AND R. DE WOLF, *Quantum algorithms for element distinctness*, in IEEE Conference on Computational Complexity, 2001, pp. 131–137.
- [7] M. ETTINGER AND P. HØYER, *On quantum algorithms for noncommutative hidden subgroups*, Adv. in Appl. Math., 25 (2000), pp. 239–251.
- [8] M. ETTINGER, P. HØYER, AND E. KNILL, *Hidden subgroup states are almost orthogonal*.
- [9] M. GRIGNI, L. J. SCHULMAN, M. VAZIRANI, AND U. V. VAZIRANI, *Quantum mechanical algorithms for the nonabelian hidden subgroup problem*, in ACM Symposium on Theory of Computing, 2001, pp. 68–74.

- [10] S. HALLGREN, A. RUSSELL, AND A. TA-SHMA, *Normal subgroup reconstruction and quantum computation using group representations*, in ACM Symposium on Theory of Computing, 2000, pp. 627–635.
- [11] P. HØYER, *personal communication*, 2003.
- [12] I. M. ISAACS AND D. S. PASSMAN, *Groups with representations of bounded degree*, *Canad. J. Math.*, 16 (1964), pp. 299–309.
- [13] G. IVANYOS, F. MAGNIEZ, AND M. SANTHA, *Efficient quantum algorithms for some instances of the non-abelian hidden subgroup problem*, *Internat. J. Found. Comput. Sci.*, 14 (2003), pp. 723–739.
- [14] A. KITAEV, *Quantum measurements and the abelian stabilizer problem*, arXiv:quant-ph/9511026.
- [15] G. KUPERBERG. `dhpsim.py`, included with the source of arXiv:quant-ph/0302112.
- [16] C. MOORE, D. ROCKMORE, AND A. RUSSELL, *Generic quantum fourier transforms*, in Proceedings of the 15th Annual ACM-SIAM Symposium on Discrete Algorithms, SIAM, Philadelphia, 2004, pp. 778–787.
- [17] M. MOSCA AND C. ZALKA, *Exact quantum fourier transforms and discrete logarithm algorithms*, *Int. J. Quantum Inf.*, 2 (2004), pp. 91–100.
- [18] M. A. NIELSEN AND I. L. CHUANG, *Quantum Computation and Quantum Information*, Cambridge University Press, Cambridge, UK, 2000.
- [19] O. REGEV, *A subexponential time algorithm for the dihedral hidden subgroup problem with polynomial space*, arXiv:quant-ph/0406151.
- [20] O. REGEV, *Quantum computation and lattice problems*, *SIAM J. Comput.*, 33 (2004), pp. 738–760.
- [21] J.-P. SERRE, *Linear Representations of Finite Groups*, Graduate Texts in Math. 42, Springer-Verlag, New York, 1977.
- [22] P. W. SHOR, *Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer*, *SIAM J. Comput.*, 26 (1997), pp. 1484–1509.
- [23] D. R. SIMON, *On the power of quantum computation*, *SIAM J. Comput.*, 26 (1997), pp. 1474–1483.
- [24] W. VAN DAM, S. HALLGREN, AND L. IP, *Quantum algorithms for some hidden shift problems*, in Proceedings of the 14th Annual ACM-SIAM Symposium on Discrete Algorithms, Baltimore, MD, 2003, SIAM, Philadelphia, 2003, pp. 489–498.

A BETTER-THAN-GREEDY APPROXIMATION ALGORITHM FOR THE MINIMUM SET COVER PROBLEM*

REFAEL HASSIN[†] AND ASAF LEVIN[‡]

Abstract. In the weighted set-cover problem we are given a set of elements $E = \{e_1, e_2, \dots, e_n\}$ and a collection \mathcal{F} of subsets of E , where each $S \in \mathcal{F}$ has a positive cost c_S . The problem is to compute a subcollection SOL such that $\bigcup_{S \in SOL} S_j = E$ and its cost $\sum_{S \in SOL} c_S$ is minimized. When $|S| \leq k \forall S \in \mathcal{F}$ we obtain the weighted k -set cover problem. It is well known that the greedy algorithm is an H_k -approximation algorithm for the weighted k set cover, where $H_k = \sum_{i=1}^k \frac{1}{i}$ is the k th harmonic number, and that this bound is exact for the greedy algorithm for all constant values of k . In this paper we give the first improvement on this approximation ratio for all constant values of k . This result shows that the greedy algorithm is not the best possible for approximating the weighted set cover problem. Our method is a modification of the greedy algorithm that allows the algorithm to regret.

Key words. greedy algorithm, approximation algorithms, set cover problem

AMS subject classifications. 68W25, 90C27

DOI. 10.1137/S0097539704444750

1. Introduction. In the weighted set-cover problem we are given a set of elements $E = \{e_1, e_2, \dots, e_n\}$ and a collection \mathcal{F} of subsets of E , where $\bigcup_{S \in \mathcal{F}} S = E$ and each $S \in \mathcal{F}$ has a positive cost c_S . The goal is to compute a subcollection $SOL \subseteq \mathcal{F}$ such that $\bigcup_{S \in SOL} S = E$ and its cost $\sum_{S \in SOL} c_S$ is minimized. Such a subcollection of subsets is called a *cover*. When we consider instances of the weighted set-cover such that each S_j has at most k elements ($|S| \leq k \forall S \in \mathcal{F}$), we obtain the weighted k -set cover problem. The unweighted set cover problem and the unweighted k -set cover problem are the special cases of the weighted set cover and of weighted k -set cover, respectively, where $c_S = 1 \forall S \in \mathcal{F}$.

It is well known (see [2]) that a greedy algorithm is an H_k -approximation algorithm for the weighted k -set cover, where $H_k = \sum_{i=1}^k \frac{1}{i}$ is the k th harmonic number, and that this bound is tight even for the unweighted version of the problem (see [10, 13]). For unbounded values of k , Slavík [17] showed that the approximation ratio of the greedy algorithm for the unweighted set cover problem is $\ln n - \ln \ln n + \Theta(1)$. Feige [5] proved that unless $NP \subseteq DTIME(n^{\text{polylog } n})$ the unweighted set cover problem cannot be approximated within a factor $(1 - \epsilon) \ln n$ for any $\epsilon > 0$. Raz and Safra [16] proved that if $P \neq NP$, then for some constant c , the unweighted set cover problem cannot be approximated within a factor $c \log n$. This result shows that the greedy algorithm is an asymptotically best possible approximation algorithm for the weighted set cover problem (unless $NP \subseteq DTIME(n^{\text{polylog } n})$). The unweighted k -set cover problem is known to be NP-complete [11] and MAX SNP-hard for all $k \geq 3$ [3, 12, 14]. The hardness results obtained for the unweighted k -set cover problem do not exclude the possibility of obtaining a cH_k -approximation algorithm for some

*Received by the editors June 27, 2004; accepted for publication (in revised form) February 2, 2005; published electronically October 3, 2005.

<http://www.siam.org/journals/sicomp/35-1/44475.html>

[†]Department of Statistics and Operations Research, Tel-Aviv University, Tel-Aviv 69978, Israel (hassin@post.tau.ac.il).

[‡]Department of Statistics, The Hebrew University, Jerusalem 91905, Israel (levinas@mscc.huji.ac.il).

constant $c < 1$ for constant values of k ; however, such positive results are unknown. Another algorithm for the weighted set cover problem by Hochbaum [9] has an approximation ratio that depends on the maximum number of subsets that contain any given item. (The local-ratio algorithm of Bar-Yehuda and Even [1] has the same performance guarantee.) See Paschos [15] for a survey on these results.

Despite this bad news, which also holds for the unweighted set cover, Goldschmidt, Hochbaum, and Yu [7] modified the greedy algorithm for the unweighted set cover and showed that the resulting algorithm has a performance guarantee of $H_k - \frac{1}{6}$. Halldórsson [8] presented an algorithm based on local search that has an approximation ratio of $H_k - \frac{1}{3}$ for the unweighted k -set cover and a $(1.4 + \epsilon)$ -approximation algorithm for the unweighted 3-set cover. Duh and Fürer [4] further improved this result and presented an $(H_k - \frac{1}{2})$ -approximation algorithm for the unweighted k -set cover.

This important progress in the approximability of the unweighted k -set cover does not help to approximate the weighted k -set cover problem within a factor better than H_k . The question whether there are similar results for the weighted problems was left unanswered. A first answer was given by Fujito and Okumura [6], who presented an $H_k - \frac{1}{12}$ -approximation algorithm for the k -set cover problem where the cost of each subset is either 1 or 2. Their method is based on extending the algorithm of [7] for this case, but this approach does not extend further to the general case. In this paper we give the first positive answer for a general cost function by developing an approximation algorithm for the weighted k -set cover problem whose performance guarantee is bounded by $H_k - \frac{k-1}{8k^9}$. This bound is not tight, but it is sufficient to show that the greedy algorithm does not give the best possible performance guarantee for approximating the weighted set cover problem. Our method is based on a modification of the greedy algorithm that allows the algorithm to regret throughout its execution.

In section 2 we review the greedy algorithm for the weighted minimum k -set cover problem and review its analysis. In section 3 we present the greedy algorithm with withdrawals. We analyze its performance in section 4. We conclude the paper in section 5 by a short discussion on possible extensions and improvements of our results.

2. The greedy algorithm. In this section we review the greedy algorithm and the proof of its performance guarantee for the weighted k -set cover problem.

The greedy algorithm starts with an empty collection of subsets in the solution and no item is covered. Then, it iterates the following procedure until all items are covered.

Let w_S be the number of uncovered items in S , and the current *ratio* of S is $r_S = \frac{c_S}{w_S}$. Let S^* be a set such that r_{S^*} is minimized. The algorithm adds S^* to the collection of subsets of the solution, it defines the items of S^* as covered, and it assigns a *price* of r_{S^*} to all the items that are now covered but were uncovered before this iteration (i.e., the items that were first covered by S^*).

Chvátal [2], extending previous results of Johnson [10] and Lovász [13] for the unweighted set cover, showed that this is an H_k -approximation algorithm for the weighted k -set cover.

Chvátal's proof is the following. First, note that the cost of the greedy solution equals the sum of prices assigned to the items. Second, consider a set S that belongs to an optimal solution OPT . Then, OPT pays c_S for S . When the i th item of S was covered by the greedy algorithm, the algorithm could choose S as a feasible solution with current ratio of $\frac{c_S}{|S|-i+1}$. (This is an upper bound on the current ratio because

S has at least $|S| - i + 1$ uncovered elements at this point.) Therefore, the price assigned to the this item is at most $\frac{c_S}{|S|-i+1}$. It follows that the total price assigned to the items of S is at most $c_S \sum_{i=1}^{|S|} \frac{1}{|S|-i+1} = c_S \sum_{i'=1}^{|S|} \frac{1}{i'} \leq c_S H_k$, and therefore, the approximation ratio of the greedy algorithm is at most H_k .

3. The greedy algorithm with withdrawals. In this section we present our modification of the greedy algorithm where in each iteration we are allowed to withdraw a subset S from the solution. However, if we decide to do so, we must pick new subsets that cover all the items that were covered by S (i.e., an item that is covered in some iteration remains covered in any future iteration). Such a *withdrawal step* allows the algorithm to overcome some choices that are essential in the known bad examples for the greedy algorithm. Incorporating the withdrawal steps, we will prove an improved performance guarantee. The first step of the algorithm adds to \mathcal{F} all the subsets of every given set $S \in \mathcal{F}$. These subsets are given the same cost as S . If $S' \subseteq S_1, S_2$, where $S_1, S_2 \in \mathcal{F}$, then $c_{S'} = \min\{c_{S_1}, c_{S_2}\}$. It is clear that this addition does not change the value of a minimum cover and optimal solutions remain optimal. Moreover, given a solution to the new instance, it is easy to compute a feasible solution to the original instance with the same cost (by mapping each new subset to the original set that contains it). However, it will make the analysis of the algorithm easier, as reflected by Lemma 2. Formally, the algorithm is defined as follows.

ALGORITHM GAWW.

Let $\alpha_k = 1 - \frac{1}{k^3}$.

1. **Initialization:** $SOL := \emptyset$ [the solution collection of subsets], $U := E$ [the set of uncovered items].
 For every subset $S \in \mathcal{F}$ and every $S' \subseteq S$, add S' to \mathcal{F} with cost $c_{S'} = c_S$. If $S' \subseteq S_1, S_2$, where $S_1, S_2 \in \mathcal{F}$, then $c_{S'} = \min\{c_{S_1}, c_{S_2}\}$. Let $\mathcal{F} := \{S_1, \dots, S_n\}$ be the resulting extended collection, and denote by c_j the cost of S_j .
2. **Iteration:** While $U \neq \emptyset$ do:
 - (a) For every j , let $w_j := |S_j \cap U|$. If $w_j \neq 0$, let $r_j := \frac{c_j}{w_j}$.
 - (b) For every $S_j \in SOL$ and every subcollection $C \subseteq \mathcal{F}$ of at most k subsets such that $S_j \subseteq \bigcup_{S \in C} S$, let $w(C) := |\bigcup_{S \in C} S \cap U|$ be the number of still uncovered items that belong to the subsets in C . If $w(C) \neq 0$, let
$$r(S_j, C) := \frac{\sum_{i: S_i \in C} c_i - c_j}{w(C)}.$$
 - (c) Let j^* be an index such that r_{j^*} is minimized (in case of a tie we prefer a minimal subset), and let \tilde{j}, \tilde{C} be such that $r(S_{\tilde{j}}, \tilde{C})$ is minimized (again, in case of a tie we prefer minimal subsets).
 - (d) **Greedy step:** If $r_{j^*} \leq \frac{1}{\alpha_k} r(S_{\tilde{j}}, \tilde{C})$, then add S_{j^*} to the solution and define the price of the newly covered items as r_{j^*} . Formally, do the following:
 - i. For every $e \in S_{j^*} \cap U$, let $price(e) := r_{j^*}$.
 - ii. $U := U \setminus S_{j^*}$.
 - iii. $SOL := SOL \cup \{S_{j^*}\}$.
 - (e) **Withdrawal step:** Otherwise (i.e., $r(S_{\tilde{j}}, \tilde{C}) < \alpha_k r_{j^*}^1$), replace $S_{\tilde{j}}$ by the subsets in \tilde{C} and define the price of the newly covered items as $r(S_{\tilde{j}}, \tilde{C})$. Formally, do the following:

¹We note that in this case each $C \in \tilde{C}$ satisfies $C \cap S_{\tilde{j}} \neq \emptyset$; however, we do not use this property and therefore do not prove it.

- i. For every $e \in \bigcup_{S \in \tilde{C}} S \cap U$, let $price(e) := r(S_j, \tilde{C})$.
- ii. $U := U \setminus \bigcup_{S \in \tilde{C}} S$.
- iii. $SOL := (SOL \setminus \{S_j\}) \cup \tilde{C}$.

3. Return SOL .

In each iteration the size of U decreases until $U = \emptyset$, and therefore the number of iterations throughout the algorithm is at most n . Since each iteration takes at most $|\mathcal{F}| + |\mathcal{F}|^k$ and because k is a constant, each iteration takes polynomial time. Therefore, the greedy algorithm with withdrawals is a polynomial time algorithm that returns a feasible solution. Therefore, we establish the following lemma.

LEMMA 1. *The greedy algorithm with withdrawals is a polynomial time algorithm for every constant value of k .*

In the next section we analyze its performance.

4. The analysis of algorithm GAWW. In this section we prove the main theorem of this paper, that for all constant values of k , GAWW guarantees a better approximation ratio than the greedy algorithm.

LEMMA 2. *We can assume without loss of generality (w.l.o.g.) that the sets in OPT are disjoint.*

Proof. The claim follows from the initialization step of GAWW in which it adds to \mathcal{F} for each subset $S \in \mathcal{F}$ with cost c_S all the subsets of S , with the same cost c_S . \square

LEMMA 3. *When analyzing the worst-case performance of GAWW, we can assume w.l.o.g. that each set of OPT has exactly k elements.*

Proof. Assume that OPT contains sets with less than k elements. We construct a new instance by extending E with a set of *dummy elements*. We extend each subset of OPT with less than k elements using some dummy elements to create a new subset with the same cost. We also add to \mathcal{F} the set $\{e\}$ for any dummy element e . All these new singleton subsets have zero cost. GAWW will first choose a full cover of the dummy elements and then continue like it acts on the original instance. However, the new instance has an optimal solution with disjoint k -sets. \square

LEMMA 4. *In each step of GAWW the sets in SOL are disjoint.*

Proof. We prove the claim by induction. The claim clearly holds when SOL is empty. It is sufficient to prove that in each step GAWW adds subsets that are disjoint to the other subsets in SOL .

- Assume that the current step is a greedy step that adds a subset X . \mathcal{F} contains all the subsets of X with costs of at most c_X . Let $X' = X \setminus \bigcup_{S \in SOL, S \neq X} S$. If $X' \neq X$, then because X' has at most the ratio of X , and GAWW prefers to use a minimal subset, we obtain a contradiction. Therefore, X is disjoint to all the prior subsets in SOL .

- Assume that the current step is a withdrawal step that withdraws a subset X and inserts $C = \{X_1, X_2, \dots, X_l\}$. Let SOL be the family of subsets in the solution before the withdrawal step took place. Recall that for all i , \mathcal{F} contains all the subsets of X_i with a cost at most c_{X_i} . Let $X'_i = (X_i \setminus (\bigcup_{S \in SOL, S \neq X} S)) \setminus \bigcup_{j \neq i} X_j$. If there exists i such that $X'_i \neq X_i$, then let C' be as C but with X'_i replacing X_i . Because the withdrawal step that withdraws X and inserts C' has at most the same ratio as the withdrawal step that withdraws X and inserts C , and GAWW prefers to use minimal subsets, we obtain a contradiction. Therefore, every X_i , $i = 1, \dots, l$, is disjoint to all the other subsets in SOL and to X_j $j \neq i$. \square

LEMMA 5. *When analyzing the performance of the greedy algorithm with withdrawals, we can assume w.l.o.g. that every X that entered the solution in some step*

of the algorithm (X may be in the final solution or it might have been withdrawn by a withdrawal step) and every $O \in OPT$, satisfy $|X \cap O| \leq 1$.

Proof. Consider an instance of the problem, and assume that the claimed property does not hold. Then we construct a new instance that satisfies the claimed property and with the same approximation ratio as the original instance. The new instance is composed of k copies of the original instance. We also add, for each $O \in OPT$, k subsets, each having exactly one element of O from each copy, and the k subsets cover the elements of the k copies of O . Then the new instance has an optimal solution that contains all these added subsets (k subsets for each original subset in OPT). GAWW applies a series of k copies of steps for each step in the original instance. Therefore, it never adds one of the new subsets but it adds only subsets that belong to a single copy of the original instance. The cost of the obtained solution is k times the cost of the resulting solution in the original instance. Therefore, the ratio in this new instance is the same as the old instance. However, the claimed property holds in the new instance. \square

LEMMA 6. *The cost of SOL is equal to $\sum_{e \in E} price(e)$, the sum of prices assigned to the elements.*

Proof. The claim clearly holds during initialization phase. Throughout an iteration we partition the increase in the cost of SOL among the new covered elements. The claim follows because the price of each element is set exactly once. \square

LEMMA 7. *Assume that $X \in SOL$ in some step of GAWW and that $x_1 \in X$ is the last element of X that was covered by GAWW. Then $price(x_1) \geq price(x_i)$ for all $x_i \in X$.*

Proof. We show a stronger claim: let x_l, x_{l-1}, \dots, x_1 be the elements of X according to the order that GAWW covers the elements, then $price(x_1) \geq price(x_2) \geq \dots \geq price(x_l)$. Suppose that X entered SOL in a greedy step. Since GAWW prefers to use a minimal subset, we conclude that all the elements of X were covered for the first time by X and all of them have the same price.

Suppose that X entered SOL in a withdrawal step that withdraws Y and inserts $C = \{X = X_1, \dots, X_l\}$. If $X \subseteq Y$, then the claim can be assumed to hold by an inductive argument. Therefore, w.l.o.g. we assume that $X \setminus Y \neq \emptyset$. Note that the elements of $X \setminus Y$ were given a common price. Then using an inductive argument it is sufficient to show that $price(x_1) \geq price(x)$ for every $x \in X \cap Y$.

We now argue that $r(Y, C)$, the ratio associated with the withdrawal step that withdraws Y and inserts C , is larger than the ratio at the step in which the algorithm inserted Y .

- Suppose that Y entered in a greedy step. Using Lemma 4,

$$\begin{aligned} r(Y, C) &= \frac{\sum_{i=1}^l c_{X_i} - c_Y}{\sum_{i=1}^l |X_i \setminus Y|} = \frac{\sum_{i=1}^l \left(c_{X_i} \frac{|X_i \setminus Y| + |X_i \cap Y|}{|X_i|} - c_Y \frac{|X_i \cap Y|}{|Y|} \right)}{\sum_{i=1}^l |X_i \setminus Y|} \\ &\geq \frac{\sum_{i=1}^l \frac{c_{X_i}}{|X_i|} |X_i \setminus Y|}{\sum_{i=1}^l |X_i \setminus Y|} \geq \min \left\{ \frac{c_{X_i}}{|X_i|} : i = 1, \dots, l \right\} \geq \frac{c_Y}{|Y|}. \end{aligned}$$

The first and third inequalities follow from $\frac{c_{X_i}}{|X_i|} \geq \frac{c_Y}{|Y|}$, which holds since Y was chosen instead of X_i and both Y and X_i were legal at the step in which GAWW inserts Y . The second inequality follows because the average is at least the minimum value.

- Suppose that Y entered SOL in a withdrawal step that withdraws a subset Z and inserts $C' = \{Y = Y_1, \dots, Y_{l'}\}$. We first prove that $r(Y, C) \geq r(Z, C')$. The proof

will assume the opposite, that $r(Y, C) < r(Z, C')$, and lead to a contradiction. Note that both ratios $r(Y, C)$ and $r(Z, C')$ refer to the time of the withdrawal that replaced Z by C' .

Consider the step in which C' replaced Z , and consider the “withdrawal step” that withdraws Z and inserts $C \cup C' \setminus \{Y\}$. (This step may be illegal because it may insert more than k subsets.) By Lemma 4, the subsets in $C' \setminus \{Y\}$ and C are pairwise disjoint. We use this last fact to show that the number of uncovered elements in $C \cup C' \setminus \{Y\}$ is $\sum_{S \in C \cup C'} |S| - |Y| - |Z|$. To see this, note that each element of $Y \cap Z$ is counted twice in $\sum_{S \in C \cup C'} |S|$ (once in C and once in C') and therefore zero times in total, each element of $Z \setminus Y$ is counted once in C' in $\sum_{S \in C \cup C'} |S|$ and therefore zero times in total, and the other elements are counted exactly once. Therefore,

$$\begin{aligned} r(Z, C \cup C' \setminus \{Y\}) &= \frac{\sum_{S \in C \cup C'} c_S - c_Y - c_Z}{\sum_{S \in C \cup C'} |S| - |Y| - |Z|} \\ &= \frac{\sum_{S \in C} c_S - c_Y}{\sum_{S \in C} |S| - |Y|} \left(\sum_{S \in C} |S| - |Y| \right) + \frac{\sum_{S \in C'} c_S - c_Z}{\sum_{S \in C'} |S| - |Z|} \left(\sum_{S \in C'} |S| - |Z| \right)}{\sum_{S \in C \cup C'} |S| - |Y| - |Z|} \\ &= \frac{r(Y, C) \left(\sum_{S \in C} |S| - |Y| \right) + r(Z, C') \left(\sum_{S \in C'} |S| - |Z| \right)}{\left(\sum_{S \in C} |S| - |Y| \right) + \left(\sum_{S \in C'} |S| - |Z| \right)} \\ &< r(Z, C'). \end{aligned}$$

The inequality follows by the assumption $r(Y, C) < r(Z, C')$. We now argue that there is a legal withdrawal step (that is, one that inserts at most k subsets) that withdraws Z with a ratio of at most $r(Z, C \cup C' \setminus \{Y\})$. By Lemma 4, because $|Z| \leq k$, there are at most k subsets in $C \cup C' \setminus \{Y\}$ that intersect Z . Let $A \in C \cup C' \setminus \{Y\}$ satisfy $A \cap Z = \emptyset$.² Then

$$\begin{aligned} r(Z, C \cup C' \setminus \{Y\}) &= \frac{\sum_{S \in C \cup C'} c_S - c_Y - c_Z}{\sum_{S \in C \cup C'} |S| - |Y| - |Z|} \\ &= \frac{\frac{c_A}{|A|} |A| + \sum_{S \in C \cup C' \setminus \{A\}} c_S - c_Y - c_Z}{\sum_{S \in C \cup C' \setminus \{A\}} |S| - |Y| - |Z|} \left(\sum_{S \in C \cup C' \setminus \{A\}} |S| - |Y| - |Z| \right)}{\sum_{S \in C \cup C'} |S| - |Y| - |Z|} \\ &= \frac{r_A |A| + r(Z, C \cup C' \setminus \{Y, A\}) \left(\sum_{S \in C \cup C' \setminus \{A\}} |S| - |Y| - |Z| \right)}{|A| + \left(\sum_{S \in C \cup C' \setminus \{A\}} |S| - |Y| - |Z| \right)} \\ &\geq \min\{r_A, r(Z, C \cup C' \setminus \{Y, A\})\}. \end{aligned}$$

Since GAWW could choose to add A greedily, $r_A > r(Z, C') > r(Z, C \cup C' \setminus \{Y\})$, and we conclude that $r(Z, C' \cup C \setminus \{Y, A\}) < r(Z, C')$. Repeating this argument we reach a legal withdrawal step (that inserts at most k subsets) with ratio that is better than $r(Z, C')$, and this is a contradiction.

We have shown that $r(Y, C) \geq r(Z, C')$, where each of these ratios is defined for the step that implemented the respective withdrawal. To complete the proof, we note that after Y is added to SOL and the withdrawal step that withdraws Y and inserts C becomes feasible, $r(Y, C)$ may increase only because the number of uncovered elements

²It is possible to show that in this case $A \in C$. However, we do not use this property and therefore we do not prove it.

of Y is nonincreasing. Thus, for every $x \in X \cap Y$, $price(x_1) = r'(Y, C) > r(Y, C) = price(x)$, where r' refers to the step that inserted X and r refers to the step that inserted Y . \square

We picture OPT as a matrix, where each column contains a subset $S_j \in OPT$. By Lemma 2 each element appears exactly once in this matrix, and by Lemma 3 each column has exactly k elements. We sort the elements of S_j according to the order they were covered by GAWW, so that the i th element of S_j that is covered by the greedy algorithm with withdrawals is in the i th row of the matrix. We break ties arbitrarily. By Lemmas 2 and 3, this representation is well defined.

For a set $X \in SOL$, denote by $COL(X)$ the submatrix defined by the set of columns where the elements of X appear.

Given $X \in SOL$, the elements of X might have been covered during different steps of GAWW (if X entered SOL in a withdrawal step and X has previously uncovered elements). Given $X \in SOL$ such that X is not contained in the last row of the matrix, we will show that the total price paid by the algorithm for all the elements in $COL(X)$ is at most $\sum_{i:S_i \in COL(X)} c_i(H_k - \frac{1}{8k^8})$. Our proof distinguishes between a set X whose elements were covered in a single step and a set X whose elements were covered in at least two steps.

LEMMA 8. *If the elements of X were covered in at least two steps, then the total price paid by GAWW for the elements in $COL(X)$ is at most $(H_k - \frac{1}{k^5+k}) \sum_{S_i \in COL(X)} c_i$.*

Proof. Assume that the element $x_1 \in X$ was covered last among the elements of X (other elements of X might have been covered during the same step). Assume that the elements of X are in columns S_1, S_2, \dots, S_l , where $X \cap S_i = \{x_i\}$, and x_i is in row $k - n_i$ of its column. (Recall that by Lemma 5, $|X \cap S_i| \leq 1$.)

Since x_1 was covered last among the elements of X , by Lemma 7 $price(x_1) \geq price(x_i) \forall i$. Since GAWW prefers minimal subsets, the step in which X was inserted to SOL is a withdrawal step. When applying the withdrawal step that inserts X into SOL , GAWW could apply a greedy step that inserts S_1 with a ratio of $\frac{c_1}{n_1+1}$. Therefore, $price(x_1) \leq \alpha_k \cdot \frac{c_1}{n_1+1} = \frac{k^3-1}{k^3} \cdot \frac{c_1}{n_1+1}$. Thus, $price(x_1) + \frac{price(x_1)}{k^3-1} \leq \frac{c_1}{n_1+1}$, and

$$\begin{aligned} price(x_1) &\leq \frac{c_1}{n_1+1} - \frac{price(x_1)}{k^3-1} \\ &\leq \frac{c_1}{n_1+1} - \frac{1}{k^3-1} \frac{1}{l} \sum_{i=1}^l price(x_i), \end{aligned}$$

where the second inequality holds because $price(x_1) \geq price(x_i) \forall i$. Therefore,

$$\begin{aligned} \sum_{i=1}^l price(x_i) &\leq \sum_{i=1}^l \frac{c_i}{n_i+1} - \frac{1}{k^3-1} \frac{1}{l} \sum_{i=1}^l price(x_i) \\ &\leq \sum_{i=1}^l \frac{c_i}{n_i+1} - \frac{1}{k^4} \sum_{i=1}^l price(x_i). \end{aligned}$$

It follows that

$$\sum_{i=1}^l price(x_i) \leq \left(1 - \frac{1}{k^4+1}\right) \sum_{i=1}^l \frac{c_i}{n_i+1}.$$

For elements $a \in S_i \setminus X$ in the $k - n_a$ row, the price of a is bounded by $\text{price}(a) \leq \frac{c_i}{n_a + 1}$. Therefore,

$$\begin{aligned} \sum_{e \in \text{COL}(X)} \text{price}(e) &= \sum_{e \in \text{COL}(X) \setminus X} \text{price}(e) + \sum_{i=1}^l \text{price}(x_i) \\ &\leq \sum_{i=1}^l \left(H_k - \frac{1}{n_i + 1} \right) c_i + \sum_{i=1}^l \left(1 - \frac{1}{k^4 + 1} \right) \frac{c_i}{n_i + 1} \\ &\leq \sum_{i=1}^l \left(H_k - \frac{1}{k^5 + k} \right) c_i, \end{aligned}$$

where the last inequality holds because $n_i + 1 \leq k$. \square

LEMMA 9. *Assume that X is not contained in the last row. If the elements of X were covered in a single step, then the total price paid by the algorithm for all the elements in $\text{COL}(X)$ is at most $(H_k - \frac{1}{8k^8}) \sum_{S_i \in \text{COL}(X)} c_i$.*

Proof. Assume that the elements of X are in columns S_1, S_2, \dots, S_l , where $X \cap S_i = \{x_i\}$, and x_i is in row $k - n_i$ of its column. By assumption, $\text{price}(x_i) = \frac{c_i}{l} \forall i$.

• Assume that $\text{price}(x_1) = \frac{c_X}{l} \leq c_1 \left(\frac{1}{n_1 + 1} - \frac{1}{4k^5} \right)$. Partition $\{2, 3, \dots, l\}$ into A and B such that $j \in A$ iff $c_j \left(\frac{1}{n_j + 1} - \frac{1}{8k^7} \right) \geq \frac{c_X}{l}$. Since $\frac{1}{n_j + 1} - \frac{1}{8k^7} \geq \frac{1}{k} - \frac{1}{8k^7} \geq \frac{1}{2k}$, it follows that $\frac{c_j}{2k} \leq \frac{c_X}{l}$ for $j \in B$. Therefore,

$$\sum_{j \in B} \frac{c_j}{8k^8} \leq \frac{1}{4k^7} \sum_{j \in B} \frac{c_X}{l} \leq \frac{c_X}{4lk^6} \leq \frac{c_1}{4k^6} \left(\frac{1}{n_1 + 1} - \frac{1}{4k^5} \right) \leq \frac{c_1}{4k^6},$$

where the second inequality follows since $|B| \leq |X| \leq k$, the third inequality follows by the assumption on x_1 , and the last inequality follows because $n_1 + 1 \geq 1$. We conclude that

$$\begin{aligned} &\sum_{e \in \text{COL}(X)} \text{price}(e) \\ &= \sum_{e \in \text{COL}(X) \setminus X} \text{price}(e) + \sum_{i=1}^l \text{price}(x_i) \\ &\leq \sum_{i=1}^l \left(H_k - \frac{1}{n_i + 1} \right) c_i + \sum_{i \in A} \left(\frac{1}{n_i + 1} - \frac{1}{8k^7} \right) c_i + \sum_{i \in B} \frac{c_i}{n_i + 1} + \left(\frac{1}{n_1 + 1} - \frac{1}{4k^5} \right) c_1 \\ &\leq \sum_{i=1}^l \left(H_k - \frac{1}{n_i + 1} \right) c_i + \sum_{i \in A} \left(\frac{1}{n_i + 1} - \frac{1}{8k^7} \right) c_i + \sum_{i \in B} \left(\frac{1}{n_i + 1} - \frac{1}{8k^8} \right) c_i + \left(\frac{1}{n_1 + 1} - \frac{1}{4k^5} + \frac{1}{4k^6} \right) c_1 \\ &\leq \left(H_k - \frac{1}{8k^8} \right) \sum_{i=1}^l c_i. \end{aligned}$$

• Assume that for every i $\text{price}(x_i) = \frac{c_X}{l} \geq c_i \left(\frac{1}{n_i + 1} - \frac{1}{4k^5} \right)$. Recall that the elements of X were covered in a single step, and not all of them are in the last row. Therefore, there exists an element $y \in \text{COL}(X)$ that was covered after X . Let y be the first element to be covered after X , and w.l.o.g. assume that $y \in S_1$.

When GAWW inserted X , it could also choose S_1 with a ratio of $\frac{c_1}{n_1 + 1}$. Therefore, $\frac{c_X}{l} \leq \frac{c_1}{n_1 + 1}$. Hence, by assumption, $c_i \left(\frac{1}{n_i + 1} - \frac{1}{4k^5} \right) \leq \frac{c_1}{n_1 + 1}$, $i = 1, \dots, l$, so that

$c_i \left(1 - \frac{1}{4k^4}\right) \leq c_i \left(1 - \frac{n_i+1}{4k^5}\right) \leq c_1 \frac{n_i+1}{n_1+1}$. Therefore, for $i = 1, \dots, l$,

$$\begin{aligned} c_i &\leq c_1 \frac{n_i + 1}{n_1 + 1} \left(1 + \frac{1}{4k^4 - 1}\right) \\ &\leq c_1 \left(\frac{n_i + 1}{n_1 + 1} + \frac{k}{4k^4 - 1}\right) \\ &\leq c_1 \left(\frac{n_i + 1}{n_1 + 1} + \frac{1}{3k^3}\right). \end{aligned}$$

When GAWW covered y it could use a withdrawal step that withdraws X and inserts S_1, S_2, \dots, S_l . Therefore,

$$\begin{aligned} price(y) &\leq \frac{1}{\alpha_k} \frac{\sum_{i=1}^l c_i - c_X}{\sum_{i=1}^l n_i} \\ &\leq \frac{1}{\alpha_k} \left(\frac{c_1 \sum_{i=1}^l \frac{n_i+1}{n_1+1} - c_X}{\sum_{i=1}^l n_i} + \frac{c_1}{3k^2}\right) \\ &= \frac{1}{\alpha_k} \left(\frac{c_1}{n_1+1} \left(1 + \frac{l}{\sum_{i=1}^l n_i}\right) - \frac{c_X}{\sum_{i=1}^l n_i} + \frac{c_1}{3k^2}\right) \\ &\leq \frac{c_1}{\alpha_k} \left(\frac{1}{n_1+1} + \frac{l}{4k^5 \sum_{i=1}^l n_i} + \frac{1}{3k^2}\right) \\ &\leq \frac{c_1}{\alpha_k} \left(\frac{1}{n_1+1} + \frac{1}{2k^2}\right) \\ &= \frac{c_1}{n_1+1} + \frac{c_1}{(k^3-1)(n_1+1)} + \frac{c_1 k^3}{2k^2(k^3-1)} \\ &\leq c_1 \left(\frac{1}{n_1+1} + \frac{9}{13k^2}\right) \\ &\leq c_1 \left(\frac{1}{n_1} - \frac{1}{4k^2}\right), \end{aligned}$$

where the first inequality follows because we could use the above withdrawal step to cover y , the second inequality follows because $c_i \leq c_1 \left(\frac{n_i+1}{n_1+1} + \frac{1}{3k^3}\right)$ for all i and $l \leq k$ (note that $\sum_{i=1}^l n_i$ can be as small as 1), the equation follows by simple algebra, the third inequality follows because by assumption $\frac{c_1}{n_1+1} - \frac{c_1}{4k^5} \leq \frac{c_X}{l}$ and therefore $\frac{c_1}{n_1+1} \frac{l}{\sum_{i=1}^l n_i} - \frac{c_X}{\sum_{i=1}^l n_i} \leq \frac{lc_1}{4k^5 \sum_{i=1}^l n_i}$, the fourth inequality follows because $\frac{1}{4k^4} + \frac{1}{3k^2} \leq \frac{1}{2k^2}$ for $k \geq 3$, the second equation follows by substituting $\alpha_k = 1 - \frac{1}{k^3}$, the fifth inequality follows because $n_1+1 \geq 2$ and for all values of $k \geq 3$ $\frac{1}{2(k^3-1)} + \frac{k}{2(k^3-1)} \leq \frac{9}{13k^2}$, and the last inequality follows because $n_1 \leq k - 1$. We conclude that

$$\begin{aligned} &\sum_{e \in COL(X)} price(e) \\ &= \sum_{e \in COL(X) \setminus (X \cup \{y\})} price(e) + \sum_{i=1}^l price(x_i) + price(y) \end{aligned}$$

$$\begin{aligned}
&\leq \left(H_k - \frac{1}{n_1+1} - \frac{1}{n_1}\right) c_1 + \sum_{i=2}^l \left(H_k - \frac{1}{n_i+1}\right) c_i + \sum_{i=1}^l \text{price}(x_i) + \left(\frac{1}{n_1} - \frac{1}{4k^2}\right) c_1 \\
&\leq \sum_{i=1}^l \left(H_k - \frac{1}{n_i+1}\right) c_i + \sum_{i=1}^l \text{price}(x_i) - \frac{1}{4k^2} \frac{c_1}{n_1+1} \\
&\leq \sum_{i=1}^l \left(H_k - \frac{1}{n_i+1}\right) c_i + \sum_{i=1}^l \text{price}(x_i) - \frac{1}{4k^2} \frac{1}{l} \sum_{i=1}^l \text{price}(x_i) \\
&\leq \sum_{i=1}^l \left(H_k - \frac{1}{n_i+1}\right) c_i + \left(1 - \frac{1}{4k^3}\right) \sum_{i=1}^l \text{price}(x_i) \\
&\leq \sum_{i=1}^l \left(H_k - \frac{1}{4k^4}\right) c_i,
\end{aligned}$$

where the first inequality follows because $\text{price}(y) \leq c_1 \left(\frac{1}{n_1} - \frac{1}{4k^2}\right)$, the second inequality follows because $n_1 + 1 \geq 1$, the third inequality follows because $\text{price}(x_i) = \text{price}(x_1) \leq \frac{c_1}{n_1+1} \forall i$, the fourth inequality follows because $l \leq k$, and the last inequality follows because $\text{price}(x_i) \leq \frac{c_i}{n_i+1}$ and therefore $(1 - \frac{1}{4k^3}) \text{price}(x_i) \leq (1 - \frac{1}{4k^3}) \frac{c_i}{n_i+1} \leq c_i \left(\frac{1}{n_i+1} - \frac{1}{4k^4}\right)$. \square

The following lemma follows from the fact that when we cover an element we pay a price that is at most the price paid for this element by the greedy algorithm.

LEMMA 10. *For every subset $S \in OPT$, the total price of the elements that belong to S is at most $H_k c_S$.*

THEOREM 11. *The approximation ratio of the greedy algorithm with withdrawals is at most $H_k - \frac{k-1}{8k^9}$. The time complexity of the algorithm is polynomial for every constant value of k .*

Proof. The claim on the time complexity of GAWW follows by Lemma 1. It remains to show the approximation ratio of GAWW. We consider the solution SOL returned by GAWW. Denote by Y the elements of the last row, and consider the following column multiset $MS = \{S \in COL(X) : X \in SOL, X \not\subseteq Y\}$. For a copy of a column S that appears in MS as part of $COL(X)$, we associate a cost of $c_S(H_k - \frac{1}{8k^8})$. By Lemmas 8 and 9, the total price of the elements in MS , where each element is counted the number of times its column appears in MS , is at most the total associated cost of the columns in MS . By Lemma 5, the number of copies of each column in this collection is at least $k - 1$ and at most k . For each column S that appears in MS $k - 1$ times, we add an additional copy of it associated with a cost of $H_k c_S$. By Lemma 10, the total price of the elements in MS , where each element is counted the number of times its column appears in MS , is at most the total associated cost of the columns in MS . Since MS has exactly k copies of each column, by Lemma 6, it suffices to show that the average associated cost of each column S is at most $(H_k - \frac{k-1}{8k^9}) c_S$. The average cost of the copies in the resulting multiset is at most $c_S \left(\frac{1}{k} H_k + \frac{k-1}{k} (H_k - \frac{1}{8k^8})\right) = c_S \left(H_k - \frac{k-1}{8k^9}\right)$, and therefore $H_k - \frac{k-1}{8k^9}$ is an upper bound on the approximation ratio of GAWW. \square

Remark 12. The following is a bad instance for the greedy algorithm with withdrawals: Assume that

$$OPT = \begin{pmatrix} X & X & X \\ Y & Y & Y \\ a & b & c \end{pmatrix},$$

where the costs of the sets in OPT (the columns of the matrix) are $c_1 = 936$, $c_2 = 1161$, and $c_3 = 1543$. Therefore $OPT = 3640$. Assume that $c_X = 936$, $c_Y = 1404$, $c_a = 774$, $c_b = 1161$, $c_c = 1543$. GAWW could pick the sets X, Y, a, b, c in this order, with a total cost of 5818. Therefore, the approximation ratio of GAWW is at least $\frac{5818}{3640} \sim 1.5984$. We note that the cost values in this example were chosen to give the worst possible ratio for any instance with the same OPT and sequence of operations chosen by the algorithm.

5. Discussion. In this paper we addressed the weighted k -set cover problem and introduced the first improvement over the greedy algorithm for any constant k . Although we are able to prove only a small improvement over the greedy algorithm, the worst example we found for $k = 3$ is with ratio of $\frac{5818}{3640}$, and the gap between these numbers is relatively large. Tightening this gap is left for future research.

In this paper we were mainly interested in proving an improvement over the greedy algorithm, and we did not try to optimize the constant and the exponent of the term $\frac{k-1}{8k^9}$.

We defined the greedy algorithm with withdrawals requiring that in each withdrawal step a single set in the current solution is withdrawn (and its elements must be covered by the new added subsets). We can generalize this algorithm and consider in each step the possibility of withdrawing at most l subsets from the current solution. This version of the algorithm is clearly polynomial for any constant l . It is natural to conjecture that the approximation ratio of the resulting algorithm is a decreasing function of l . In this paper we proved that $l = 1$ strictly improves the bound of $l = 0$ which corresponds to the greedy algorithm of Chvátal [2].

We conclude this discussion by noting that the withdrawal operation was crucial to obtain better approximation ratio. We mean that allowing the greedy algorithm to insert p subsets at each step (where p is some constant) does not improve the approximation ratio of the algorithm. To see this, note that we can take p copies of an original instance, and the new algorithm will have the same performance guarantee on the new instance as the greedy algorithm has on the original instance.

REFERENCES

- [1] R. BAR-YEHUDA AND S. EVEN, *A linear time approximation algorithm for the weighted vertex cover problem*, J. Algorithms, 2 (1981), pp. 198–203.
- [2] V. CHVÁTAL, *A greedy heuristic for the set-covering problem*, Math. Oper. Res., 4 (1979), pp. 233–235.
- [3] P. CRESCENZI AND V. KANN, *A compendium of NP optimization problems*, <http://www.nada.kth.se/theory/problemlist.html> (1995).
- [4] R. DUH AND M. FÜRER, *Approximation of k -set cover by semi local optimization*, in Proceedings of the STOC, 1997, pp. 256–264.
- [5] U. FEIGE, *A threshold of $\ln n$ for approximating set cover*, J. ACM, 45 (1998), pp. 634–652.
- [6] T. FUJITO AND T. OKUMURA, *A modified greedy algorithm for the set cover problem with weights 1 and 2*, in Proceedings of the 12th International Symposium on Algorithms and Computation, 2001, pp. 670–681.
- [7] O. GOLDSCHMIDT, D. S. HOCHBAUM, AND G. YU, *A modified greedy heuristic for the set covering problem with improved worst case bound*, Inform. Process. Lett., 48 (1993), pp. 305–310.
- [8] M. M. HALLDÓRSSON, *Approximating k -set cover and complementary graph coloring*, in Proceedings of the Integer Programming and Combinatorial Optimization Conference, 1996, pp. 118–131.
- [9] D. S. HOCHBAUM, *Approximation algorithms for the weighted set covering and node cover problems*, SIAM J. Comput., 11 (1982), pp. 555–556.
- [10] D. S. JOHNSON, *Approximation algorithms for combinatorial problems*, J. Comput. System Sci., 9 (1974), pp. 256–278.

- [11] R. M. KARP, *Reducibility among combinatorial problems*, in Complexity of Computer Computations, R. E. Miller and J. W. Thatcher, eds., Plenum Press, New York, 1972, pp. 85–103.
- [12] S. KHANNA, R. MOTWANI, M. SUDAN, AND U. VAZIRANI, *On syntactic versus computational views of approximability*, SIAM J. Comput., 28 (1998), pp. 164–191.
- [13] L. LOVÁSZ, *On the ratio of optimal integral and fractional covers*, Discrete Math., 13 (1975), pp. 383–390.
- [14] C. H. PAPADIMITRIOU AND M. YANNAKAKIS, *Optimization, approximation, and complexity classes*, J. Comput. System Sci., 43 (1991), pp. 425–440.
- [15] V. T. PASCHOS, *A survey of approximately optimal solutions to some covering and packing problems*, ACM Comput. Surveys, 29 (1997), pp. 171–209.
- [16] R. RAZ AND S. SAFRA, *A sub-constant error-probability low-degree test, and sub-constant error-probability PCP characterization of NP*, in Proceedings of the STOC, 1997, pp. 475–484.
- [17] P. SLAVÍK, *A tight analysis of the greedy algorithm for set cover*, J. Algorithms, 25 (1997), pp. 237–254.

A SUPERPOLYNOMIAL LOWER BOUND FOR A CIRCUIT COMPUTING THE CLIQUE FUNCTION WITH AT MOST (1/6) log log N NEGATION GATES*

KAZUYUKI AMANO[†] AND AKIRA MARUOKA[†]

Abstract. In this paper, we investigate the lower bound on the number of gates in a Boolean circuit that computes the clique function with a limited number of negation gates. To derive strong lower bounds on the size of such a circuit we develop a new approach by combining three approaches: the restriction applied to constant depth circuits due to Håstad, the approximation method applied to monotone circuits due to Razborov, and the boundary covering developed in the present paper. We prove that if a circuit C with at most $\lfloor (1/6) \log \log m \rfloor$ negation gates detects cliques of size $(\log m)^{3(\log m)^{1/2}}$ in a graph with m vertices, then C contains at least $2^{(1/5)(\log m)^{1/2}}$ gates. No nontrivial lower bounds on the size of such circuits were previously known, even if we restrict the number of negation gates to be a constant. Moreover, it follows from a result of Fischer [*Lect. Notes Comput. Sci.*, 33 (1974), pp. 71–82] that if one can improve the number of negation gates from $\lfloor (1/6) \log \log m \rfloor$ to $\lfloor 2 \log m \rfloor$ in the statement, then we have $P \neq NP$. We also show that the problem of lower bounding the negation-limited circuit complexity can be reduced to the one of lower bounding the maximum of the monotone circuit complexity of the functions in a certain class of monotone functions.

Key words. circuit complexity, monotone circuit, negation-limited circuit, approximation method, lower bound, clique function

AMS subject classifications. 06E30, 68Q17, 68Q25, 94C10

DOI. 10.1137/S0097539701396959

1. Introduction. There has been substantial progress in obtaining strong lower bounds on the size of restricted Boolean circuits, such as constant depth circuits or monotone circuits that compute certain functions. For example, exponential lower bounds were derived for the size of constant depth circuits computing the parity function [7] and for the size of monotone circuits computing the clique function and other functions (see [1, 2, 5, 8, 9, 10, 13]). It is natural to ask if we could use the approaches developed so far to derive strong lower bounds for a more general model. In such a generalized model, we consider circuits with a limited number of negation gates. In fact, it so far remains open to derive nontrivial lower bounds on the size of a circuit computing a certain monotone function with, say, a constant number of negation gates [14]. Fischer [6] showed that for any function f on n variables, the size of the smallest circuit computing f with an arbitrary number of NOT gates and the one with, at most, $\lceil \log(n+1) \rceil$ NOT gates are polynomially related (see also [4]). So if one can prove superpolynomial lower bounds on the size of circuits with at most $\lceil \log(n+1) \rceil$ NOT gates computing an explicit function in NP, then we have $P \neq NP$.

In this paper, we try to obtain superpolynomial lower bounds on the size of circuits computing an explicit function in NP with $O(\log \log n)$ NOT gates rather than $O(\log n)$ NOT gates. More precisely we prove the following: If a circuit C

*Received by the editors October 24, 2001; accepted for publication (in revised form) June 29, 2005; published electronically October 3, 2005. A preliminary version of this paper appeared in *Proceedings of the 23rd International Symposium on Mathematical Foundations of Computer Science (MFCS '98)*, Lect. Notes Comput. Sci. 1450, Brno, Czech Republic, 1998, pp. 399–408.

<http://www.siam.org/journals/sicomp/35-1/39695.html>

[†]Graduate School of Information Sciences, Tohoku University, Aoba 6-6-05, Aramaki, Sendai 980-8579, Japan (ama@ecei.tohoku.ac.jp, maruoka@ecei.tohoku.ac.jp).

with at most $\lfloor (1/6) \log \log m \rfloor$ NOT gates detects cliques of size $(\log m)^{3(\log m)^{1/2}}$ in a graph with m vertices, then C contains at least $2^{(1/5)(\log m)^{(\log m)^{1/2}}}$ gates (Theorem 5.1). Note that the problem of detecting a clique in a graph with m vertices will be written as a Boolean function of $n = \binom{m}{2}$ variables. We also show that the problem of lower bounding the negation-limited circuit complexity can be reduced to the one of lower bounding the maximum of the monotone circuit complexity of the functions in a certain class of monotone functions. (Theorem 3.2).

To achieve the main results, we develop a new approach by combining three approaches: the restriction applied to constant depth circuits [7], the approximation method applied to monotone circuits [10, 11], and the boundary covering developed in the present paper. A Boolean function f partitions the Boolean cube into two regions, $f^{-1}(0)$ and $f^{-1}(1)$. We can think of the boundary between the two regions, defined as the collection of pairs of vectors (w, w') such that $f(w) \neq f(w')$, and the Hamming distance between w and w' is 1. The idea of the proof of the main theorem is as follows. First, we prove in section 3 a theorem showing that the problem of proving a lower bound on the size of the negation-limited circuit computing a monotone function f can be reduced to that of proving a lower bound on the maximum over sizes of monotone circuits such that the union of boundaries of the functions computed by the monotone circuits covers the boundary of the monotone function f (Theorem 3.2). Second, we analyze carefully in section 4 the proof of Amano and Maruoka [2] that gives an exponential lower bound on the monotone circuit size of the clique function, and prove a statement saying that we still need a superpolynomial number of gates in a monotone circuit that implements even a certain small fraction of the boundary of the clique function (Theorem 4.1). Finally, we prove in section 5 a statement (Theorem 5.1) that no matter what collection of monotone functions we take to cover the boundary of the clique function, the largest fraction of the boundary covered by a monotone function in the collection is more than what is needed to apply the result (Theorem 4.1) in section 4. This is the most difficult part of the proof.

2. Preliminaries. For w in $\{0, 1\}^n$, let w_i denote the value of the i th bit of w . Let w and w' be in $\{0, 1\}^n$. We denote $w \leq w'$ if $w_i \leq w'_i$ for all $1 \leq i \leq n$, and $w < w'$ if $w \leq w'$ and $w \neq w'$. Let $\text{Ham}(w, w')$ denote the Hamming distance between w and w' , i.e., $\text{Ham}(w, w') = |\{i \in \{1, \dots, n\} \mid w_i \neq w'_i\}|$, where $|S|$ denotes the number of elements in a set S . For two Boolean functions f and g of n variables, we write $f \leq g$ if $f(w) \leq g(w)$ for all $w \in \{0, 1\}^n$.

A *Boolean circuit* is a directed acyclic graph with gate nodes (or, simply gates) and input nodes. The operation AND or OR is associated with each gate whose indegree is 2, whereas NOT is associated with each gate whose indegree is 1. A Boolean variable or a constant, namely, 0 or 1, is associated with each input node whose indegree is 0. There is one designated node in a Boolean circuit with outdegree 0, which is called the output gate. Each gate of the circuit computes a Boolean function in the obvious way and the function computed by the circuit is the function computed by the output gate. In particular, a circuit with no NOT gates is called a *monotone circuit*.

A Boolean function of n variables is *monotone* if $f(w) \leq f(w')$ holds for any $w, w' \in \{0, 1\}^n$ such that $w \leq w'$. Let \mathcal{M}^n denote the set of all monotone functions on n variables. It is well known that f is monotone if and only if f can be computed by a monotone circuit. The *size* of a circuit C , denoted $\text{size}(C)$, is the number of gates in the circuit C . The *circuit complexity* (respectively, *monotone circuit complexity*) of a function f , denoted $\text{size}(f)$ (respectively, $\text{size}_{\text{mon}}(f)$), is the size of the smallest circuit (respectively, the smallest monotone circuit) computing f . For a function f and

a positive integer t , the *circuit complexity with t limited negation* (negation limited complexity, for short) of a function f , denoted $\text{size}_t(f)$, is the size of the smallest circuit C that computes f and includes at most t NOT gates. If a function f cannot be computed with only t NOT gates, then $\text{size}_t(f)$ is undefined. For a circuit C and a gate g in C , let $C_g(w)$ denote the output of the gate g in the circuit C that has the input w . We sometimes assume that a gate g of circuit C is specified as an output gate of the circuit C . In such a case, we say the circuit computes the function $C_g(w)$, which will simply be denoted by $C(w)$. We say a gate g in a circuit C separates a pair of vectors (w, w') (or simply, a gate g separates (w, w') when no confusion arises) if $C_g(w) = 0$, $C_g(w') = 1$ and $\text{Ham}(w, w') = 1$. In particular, when g is taken to be the output gate in C , we simply say that the circuit C separates such a pair (w, w') . Similarly, when a circuit separates a pair we say the function computed by the circuit separates the pair.

Throughout this paper, the function $\log x$ denotes the logarithm base 2 of x .

3. Relationship between negation-limited and monotone circuit complexity. In this section, we explore a relationship between negation-limited circuit complexity and monotone circuit complexity for a monotone Boolean function.

DEFINITION 3.1. *Let f be a Boolean function of n variables. A boundary graph of f , denoted $G(f)$, is defined as follows: $G(f) = (V, E)$ is a directed graph with $V = \{0, 1\}^n$ and $E = \{(w, w') \mid \text{Ham}(w, w') = 1, f(w) = 0 \text{ and } f(w') = 1\}$.*

So a boundary graph of f is a graph whose edge set consists of pairs that are separated by the function f . Let $G_1 = (V, E_1)$ and $G_2 = (V, E_2)$ be two graphs on the same set V of vertices. Then the *union* $G_1 \cup G_2$ is defined to be the graph $(V, E_1 \cup E_2)$. Furthermore, we say G_1 contains G_2 , denoted by $G_1 \supseteq G_2$, if $E_1 \supseteq E_2$ holds.

THEOREM 3.2. *Let f be a monotone function of n variables. For any positive integer t ,*

$$\text{size}_t(f) \geq \min_{F' = \{f_1, \dots, f_\alpha\} \subseteq \mathcal{M}^n} \left\{ \max_{f' \in F'} \{ \text{size}_{\text{mon}}(f') \} \mid \bigcup_{f' \in F'} G(f') \supseteq G(f) \right\},$$

where $\alpha = 2^{t+1} - 1$. Here the functions f_i are not necessarily distinct, so F' could be a multiset.

This theorem shows that the problem of deriving lower bounds on the negation-limited circuit complexity of a monotone function can be reduced to the one of deriving the maximum monotone circuit complexity over the monotone functions such that the corresponding boundary graphs of the functions cover that of the original function. Intuitively, this is because any pair of vectors (w, w') separated by a function f , which is supposed to be computed by a negation-limited circuit C , belongs to the boundary of a monotone circuit obtained by restricting outputs of some NOT gates (possibly, all of the NOT gates) in C , to constants 0 or 1 appropriately and throwing away the remaining NOT gates in C . So lines where we place restriction to constants are outputs of some NOT gates inside of the circuit rather than inputs to the entire circuit as in the case of the lower bound proof for constant depth circuits.

It is worthwhile to note that the set of all variables $F' = \{x_1, \dots, x_n\}$ satisfies the condition $\bigcup_{f' \in F'} G(f') \supseteq G(f)$ for any monotone function f . Hence, if $\alpha \geq n$, that is, $t \geq \log(n + 1) - 1$, the right-hand side of the inequality in Theorem 3.2 does not give a nontrivial lower bound.

Proof of Theorem 3.2. Let f be a monotone function of n variables. Let C be the smallest circuit that computes f using no more than t NOT gates, that is, $\text{size}(C) = \text{size}_t(f)$. Without loss of generality, we assume the number of NOT gates in C is given by t . Furthermore, without loss of generality we assume that the output gate of C is not a NOT gate. Let g_1, \dots, g_t be a list of NOT gates of C arranged in the topological ordering and g_{t+1} be the output gate of the entire circuit C . For $0 \leq i \leq t$ and $u = (u_1, \dots, u_i) \in \{0, 1\}^i$, let C_u denote the subcircuit of C obtained by fixing the output of the NOT gates g_j to constant u_j for $1 \leq j \leq i$ and making the input to g_{i+1} in C the output of the circuit C_u . In particular, for the empty sequence λ , C_λ denotes the circuit obtained by making the input to g_1 in C the output of entire circuit C_λ .

Clearly, for any u of length at most t , the function computed by the circuit C_u is monotone, and the size of the circuit C_u is not greater than the size of the circuit C . Then it is easy to see that, for any $(w, w') \in \{0, 1\}^n \times \{0, 1\}^n$ separated by the circuit C , there exists $0 \leq i \leq t$ and $u \in \{0, 1\}^i$ such that the circuit C_u separates the (w, w') . This is because as such an i we can simply take i such that g_{i+1} is the first gate in the sequence (g_1, \dots, g_{t+1}) such that $C_{g_{i+1}}(w) \neq C_{g_{i+1}}(w')$, and put $u_j = g_j(w) (= g_j(w'))$ for $1 \leq j \leq i$. The number of the circuits represented as C_u for $u \in \{0, 1\}^*$ such that $|u| \leq t$ is given by $\sum_{j=0}^t 2^j = 2^{t+1} - 1 = \alpha$. Hence, denoting by f_j 's functions computed by circuit C_u 's, we have $\bigcup_{f' \in F'} G(f') \supseteq G(f)$ for $F' = \{f_1, \dots, f_\alpha\}$. Thus, since $\text{size}_t(f) (= \text{size}(C)) \geq \text{size}_{\text{mon}}(f')$ for any $f' \in F'$, the proof is completed. \square

4. Hardness of approximating clique function. The clique function, denoted $\text{CLIQUE}(m, s)$, of $m(m-1)/2$ variables is defined to take the value 1 if and only if the undirected graph on m vertices represented in the obvious way by the input contains a clique of size s . For a positive integer s_2 , a graph on m vertices is called *good* if it consists of a clique on some set of s_2 vertices and contains no other edges. Let $I(m, s_2)$ denote the set of such good graphs. For a positive integer s_1 , a graph on m vertices is called *bad* if there is a partition of the vertices into $m \bmod (s_1 - 1)$ sets of size $\lceil m/(s_1 - 1) \rceil$ and $s_1 - 1 - (m \bmod (s_1 - 1))$ sets of size $\lfloor m/(s_1 - 1) \rfloor$ such that any two vertices chosen from different sets have an edge between them, and no other edges exist. Let $O(m, s_1)$ denote the set of such bad graphs.

For $1 \leq s_1 \leq s_2 \leq m$, let $F(m, s_1, s_2)$ denote the set of all the monotone functions f of $\binom{m}{2}$ variables representing a graph G on m vertices such that the function f outputs 0 if G contains no clique of size s_1 , outputs 1 if G contains a clique of size s_2 , and outputs an arbitrary value otherwise. We remark that $F(m, s, s) = \{\text{CLIQUE}(m, s)\}$ and that if $s_1 < s_2$, then $F(m, s_1, s_2)$ consists of more than one function. For any function f in $F(m, s_1, s_2)$, the value of f is 1 for any good graph, and is 0 for any bad graph. Moreover, a good graph is minimal in the sense that removing any edge from the graph destroys the clique of size s_2 . On the other hand, a bad graph is maximal in the sense that adding any edge to the graph makes the graph contain a clique of size s_1 .

In this section, we prove the following theorem, which will be needed in the next section to prove the main theorem. This theorem says that, if $64 \leq s_1 \leq s_2$ and $s_1^{1/3} s_2 \leq m/200$, then no function in $F(m, s_1, s_2)$ can be approximated by a feasible monotone circuit.

THEOREM 4.1. *Let s_1 and s_2 be positive integers such that $64 \leq s_1 \leq s_2$ and $s_1^{1/3} s_2 \leq m/200$. Suppose that C is a monotone circuit and that the fraction of good graphs in $I(m, s_2)$ such that C outputs 1 is at least $h = h(s_2)$. Then at least one of*

the following holds:

- (i) The number of gates in C is at least $(h/2)2^{s_1^{1/3}/4}$.
- (ii) The fraction of bad graphs in $O(m, s_1)$ such that C outputs 0 is at most $2/s_1^{1/3}$.

Let $\Pr_{v \in I(m, s_2)}[E(v)]$ denote the probability of event $E(v)$ provided that the uniform distribution over $I(m, s_2)$ is assumed, and similarly for $\Pr_{u \in O(m, s_1)}[E(u)]$. Theorem 4.1 is restated as follows: Assume that the conditions on the parameters described in the theorem are satisfied. If monotone circuit C is such that

$$\Pr_{v \in I(m, s_2)}[C(v) = 1] \geq h,$$

and

$$\Pr_{u \in O(m, s_1)}[C(u) = 0] > \frac{2}{s_1^{1/3}},$$

then

$$\text{size}(C) \geq \left(\frac{h}{2}\right) 2^{s_1^{1/3}/4}.$$

The proof of Theorem 4.1 is done by employing the symmetric version of the method of approximation [2, 5, 8, 9, 13]. In particular, we use arguments similar to the proof of an exponential lower bound on the monotone circuit complexity of the clique function (Theorem 3.1 in [2]). The key to the proof is to define the approximate operations $\bar{\vee}$ (which approximates an OR gate) and $\bar{\wedge}$ (which approximates an AND gate) in terms of disjunctive normal form (DNF) and conjunctive normal form (CNF) formulas such that the size of terms and clauses in the formulas is limited appropriately. For the purpose of the arguments of the current paper, we adopt the same definition for the approximate operations as in [2] except for the values of the parameters l and r in their definitions, and follow their arguments to obtain Theorem 4.1.

In what follows we present a rough sketch of a proof of Theorem 4.1. As in [2], a monotone circuit is assumed to be converted to satisfy the following conditions: Any input of an OR gate (respectively, an AND gate) is connected to either an output of an AND gate (respectively, an OR gate) or an input node; and the output gate of the circuit is an AND gate. It is easy to see that in order to convert a monotone circuit to satisfy these conditions, we need to at most double the size of the circuit.

A DNF formula is a disjunction of conjunctions of input variables and each conjunction of a DNF formula is called a *term*. A CNF formula is a conjunction of disjunctions of input variables and each disjunction of a CNF formula is called a *clause*. Let t be a term or a clause. The *endpoint set* of t is a set of all endpoints of the edges corresponding to variables in t . The *size* of t is defined to be the cardinality of the endpoint set of t . The approximate operations $\bar{\vee}$ and $\bar{\wedge}$ are defined as follows:

$\bar{\vee}$: Let f_1^D and f_2^D be two functions, represented by monotone DNF formulas, feeding into an $\bar{\vee}$ gate. $f_1^D \bar{\vee} f_2^D$ is the CNF formula obtained by transforming the monotone DNF formula $f_1^D \vee f_2^D$ into the monotone CNF formula and then taking away all the clauses whose size exceeds r .

$\bar{\wedge}$: Let f_1^C and f_2^C be two functions, represented by monotone CNF formulas, feeding into an $\bar{\wedge}$ gate. $f_1^C \bar{\wedge} f_2^C$ is the DNF formula obtained by transforming the monotone CNF formula $f_1^C \wedge f_2^C$ into the monotone DNF formula and then taking away all the terms whose size exceeds l .

In what follows, AND and OR gates are also written as \wedge and \vee gates, respectively. Given a monotone circuit C , the circuit obtained by replacing all \vee and \wedge gates in C by $\bar{\vee}$ and $\bar{\wedge}$ gates, respectively, is denoted by \bar{C} , which will be called the approximator circuit corresponding to C . The approximate operations and the approximator circuits are introduced to derive good lower bounds on the size of circuits computing a certain function. Theorem 4.1 derives a lower bound on the size of monotone circuits approximately computing the clique function. Its proof, based on the approximation method, is as follows. First, we show that the number of good and bad graphs that are classified incorrectly by an approximator circuit is large (Lemma 4.3). More precisely, for an approximator circuit \bar{C} arbitrarily given, \bar{C} outputs 0 for a large number of good graphs or yields 1 for a large number of bad graphs. Second, we show that the number of good and bad graphs for which a usual gate and the corresponding approximate gate behave differently is small. More precisely, it is only for a small number of bad graphs that \vee gate outputs 0 and $\bar{\vee}$ gate yields 1 (Lemma 4.4), whereas it is only for a small number of good graphs that \wedge gate outputs 1 and $\bar{\wedge}$ gate yields 0 (Lemma 4.5). Recall that, in general, the usual gates and the corresponding approximate gates behave differently because long clauses or long terms are taken away when defining the approximate operations based on the formulas. Finally, since for each good or bad graph classified incorrectly by an approximator circuit there exist an approximate gate in \bar{C} that behaves differently from the corresponding usual gate on that graph, the number of approximate gates that compensate for a large number of good or bad graphs misclassified by the entire circuit must be large (Theorem 4.1).

Choose $l = \lfloor s_1^{1/3}/4 \rfloor$ and $r = \lfloor 30s_1^{1/3} \rfloor$. Put $w = m \bmod (s_1 - 1)$. A simple calculation shows the following.

FACT 4.2. $|I(m, s_2)| = (m!)/(s_2!(m - s_2)!) \text{ and}$

$$|O(m, s_1)| = \frac{m!}{(\lceil m/(s_1 - 1) \rceil!)^w (\lfloor m/(s_1 - 1) \rfloor!)^{s_1 - 1 - w} w! (s_1 - 1 - w)!}.$$

We proceed to the technical parts of the proof. Although the arguments are analogous to that presented in [2], we give proofs here to make this paper self-contained.

LEMMA 4.3. *Let C be a monotone circuit. An approximator circuit \bar{C} outputs identically 0 or the fraction of bad graphs in $O(m, s_1)$ such that \bar{C} outputs 1 is at least $1 - s_1^{-1/3}$.*

Proof. Let \bar{f} be the output of an approximator circuit \bar{C} . Because of the assumption that the output gate of the approximator circuit is an AND gate, \bar{f} can be represented by a monotone DNF formula consisting of terms of size at most l . If \bar{f} is identically 0, then the first conclusion holds. If not, then there is a term t whose size is at most l such that $\bar{f} \geq t$ holds. In what follows, bad graphs are represented as one-to-one mapping from the vertex set to $\{(1, 1), \dots, (1, \lceil m/(s_1 - 1) \rceil), \dots, (s_1 - 1, 1), \dots, (s_1 - 1, \lfloor m/(s_1 - 1) \rfloor)\}$, so there are many mappings corresponding to one bad graph. Such a mapping specifies a bad graph in the obvious way: Two vertices in the graph have an edge between them if and only if the mapping assigns pairs to the vertices with different first components. The function in question will be estimated in terms of the ratio of the corresponding mappings. It is easy to see that the ratio of mappings that satisfy the condition that there is a variable x in the term t such that the two vertices incident to x are assigned a pair with the same first component, i.e.,

the term t outputs 0 on the bad graphs specified by such mappings, is at most

$$\frac{l(l-1)}{2} \frac{\lceil m/(s_1-1) \rceil}{m} < \frac{s_1^{2/3}}{32} \frac{2}{s_1} < s_1^{-1/3}.$$

This completes the proof. \square

LEMMA 4.4. *Suppose \vee gate and ∇ gate are given as inputs the same monotone formulas such that the size of any term in the formulas is at most l . The number of bad graphs in $O(m, s_1)$ for which the OR and ∇ gates produce different outputs (the OR gate produces 0, whereas the ∇ gate produces 1) is at most*

$$(4.1) \quad \frac{(m/s_1^{1/6})^{r+1}(m-r-1)!}{(\lceil m/(s_1-1) \rceil!)^w (\lfloor m/(s_1-1) \rfloor!)^{s_1-1-w} w!(s_1-1-w)!},$$

where $w = m \bmod (s_1 - 1)$.

Proof. Let f_1^D and f_2^D denote monotone formulas, such that the size of any term in the formulas is at most l . Let $f_1^D \vee f_2^D$ and $f_1^D \nabla f_2^D$ be denoted by f^D and f^C , respectively. Let t_1, \dots, t_q be the complete list of the terms in f^D . We shall count the number of bad graphs u such that both $f^D(u) = 0$ and $f^C(u) = 1$ hold. As in the proof of Lemma 4.3, bad graphs are represented as the mappings described there. Since each bad graph in $O(m, s_1)$ is represented exactly as $(\lceil m/(s_1-1) \rceil!)^w (\lfloor m/(s_1-1) \rfloor!)^{s_1-1-w} w!(s_1-1-w)!$ mappings as in the proof of Lemma 4.3, it suffices to show that the number of mappings corresponding to the bad graphs described in the lemma is at most $(m/s_1^{1/6})^{r+1}(m-r-1)!$. In order to count such mappings we will count the number of ways of choosing one or two vertices in a certain manner repeatedly from the variables in t_1, \dots, t_q so that the corresponding bad graph u satisfies the conditions $f^D(u) = 0$ and $f^C(u) = 1$.

Suppose that we somehow already assigned distinct pairs of integers to endpoints of variables from terms t_1, \dots, t_{i-1} so as to make all these terms take the value 0 and that we proceed to the term t_i . We now typically choose one or two vertices from the endpoints of variables from t_i in the way described below and then proceed to the next term t_{i+1} .

We first consider two extreme cases. If there is a variable in t_i already assigned 0 by the partial assignment, we skip to the next term t_{i+1} . The other extreme case occurs when all the variables in t_i are, so far, assigned 1. In this case the term t_i will never take value 0, hence we do not need to consider the case.

If neither of these extreme cases happens, choose a variable from the term t_i such that at least one of the vertices associated with the variable is not assigned a pair of integers. There are two cases to consider: If exactly one of the vertices has been assigned, then assign a pair to the remaining vertex whose first component is identical to the first component of the pair of integers associated with the variable so that the variable takes the value 0. In this case, there are at most $\lfloor m/(s_1-1) \rfloor \leq m/(s_1-1)$ ways of assigning the pairs of integers to the vertex. On the other hand, if both of the vertices have not been chosen, assign to these vertices pairs of integers with their first components being the same so that the variable associated with two vertices takes the value 0. So, for the two vertices, there are at most $(s_1-1)(\lceil m/(s_1-1) \rceil)(\lfloor m/(s_1-1) \rfloor - 1) \leq 2m^2(s_1-1)$ ways of assigning the pairs of integers.

Let k be the number of variables in term t_i such that exactly one of the vertices corresponding to the variables is assigned a pair of integers so far. Then there exist at most $l(l-1)/2 - k$ variables in term t_i such that none of the vertices associated

with the variables has been assigned a pair of integers so far. So the number of ways of choosing an unassigned vertex in the endpoints of variables in t_i and assigning a pair of integers to the chosen vertex is at most

$$(4.2) \quad \begin{aligned} & \max_{0 \leq k \leq l(l-1)/2} \{k(m/(s_1 - 1)) + \sqrt{(l(l-1)/2 - k)(2m^2/(s_1 - 1))}\} \\ & \leq (l(l-1)/2)(m/(s_1 - 1)) + \sqrt{(l(l-1)/2)(2m^2/(s_1 - 1))}. \end{aligned}$$

This is because doing something to two vertices in i ways can be regarded as doing something to a vertex appropriately in \sqrt{i} ways twice successively. Recall that $l = \lfloor s_1^{1/3}/4 \rfloor$. We have that

$$(4.2) \leq \frac{s_1^{2/3}}{32} \frac{2m}{s_1} + \sqrt{\frac{s_1^{2/3}}{32} \frac{4m^2}{s_1}} < \frac{m}{32s_1^{1/3}} + \frac{m}{2s_1^{1/6}} < \frac{m}{s_1^{1/6}}.$$

By the definition of ∇ gate, a bad graph u corresponding to a mapping specified in this way satisfies $f^D(u) = 0$ and $f^C(u) = 1$ only if there exist more than r vertices assigned to pairs of integers in the above procedure. Thus the number of mappings corresponding to such bad graphs is at most the number of the ways of assigning the $r + 1$ vertices to the pairs of integers in the manner described above multiplied by the number of ways of assigning the remaining $m - r - 1$ vertices arbitrarily to the remaining distinct pairs of integers, which is given by $(m/s_1^{1/6})^{r+1}(m - r - 1)!$. This completes the proof. \square

LEMMA 4.5. *The number of good graphs in $I(m, s_2)$ for which the AND and $\bar{\wedge}$ gates produce different outputs (the AND gate produces 1, whereas the $\bar{\wedge}$ gate produces 0) is at most*

$$\frac{(2rs_2)^{l+1}(m - l - 1)!}{s_2!(m - s_2)!}.$$

Proof. The proof is similar to that of Lemma 4.4. Suppose an AND gate and an $\bar{\wedge}$ gate are given as input for the same monotone CNF formulas, denoted f_1^C and f_2^C . Let $f^C = f_1^C \wedge f_2^C$ and $f^D = f_1^C \bar{\wedge} f_2^C$. Let c_1, \dots, c_q be the complete list of clauses in f^C . Note that, for each clause c_i , the size of c_i is at most r and so the clause c_i contains at most $r(r-1)/2$ variables. The number in question is equal to the number of good graphs v such that $f^C(v) = 1$ and $f^D(v) = 0$.

Instead of the mappings from vertices to pairs of integers in the case of Lemma 4.4, we consider one-to-one mappings from the vertex set to the set of integers $\{1, \dots, m\}$. Such a mapping is thought to specify a good graph such that the set of vertices assigned with integers from 1 to s_2 forms a clique. Since each good graph in $I(m, s_2)$ is represented as exactly $s_2!(m - s_2)!$ mappings, it suffices to show that the number of mappings corresponding to good graphs v such that $f^C(v) = 1$ and $f^D(v) = 0$ is at most $(2rs_2)^{l+1}(m - l - 1)!$.

As in the proof of Lemma 4.4 we proceed from c_1 up to c_q repeatedly by assigning one or two vertices to integers from $\{1, \dots, m\}$ so that the resulting good graph v satisfies the condition that $f^C(v) = 1$ and $f^D(v) = 0$. Suppose that we somehow already assigned distinct integers to the endpoints of variables from clauses c_1, \dots, c_{i-1} so as to make all these clauses take the value 1 and we proceed to clauses c_i . We now choose one or two vertices from the endpoints of variables from c_i in a way similar to that in the proof of Lemma 4.4 and then proceed to the next clause c_{i+1} . As in

the case of the proof of Lemma 4.4, we need only to consider the case where there remain variables in c_i such that assigning one or two endpoints, not chosen so far, of these variables makes the clause c_i take the value 1. Since the number of endpoints of vertices in c_i is at most r , the number of ways of choosing a vertex and assigning an integer in the manner described above is at most $r(s_2 - 1)$, while the number of ways of choosing two adjacent vertices and assigning integers is at most $(r(r - 1)/2)s_2(s_2 - 1)$. Thus the number of ways per a vertex is at most

$$r(s_2 - 1) + \sqrt{(r(r - 1)/2)s_2(s_2 - 1)} < 2rs_2.$$

Thus, in a way similar to the proof of Lemma 4.4, it is easily seen that the number of mappings corresponding to good graphs v such that $f^C(v) = 1$ and $f^D(v) = 0$ is at most $(2rs_2)^{l+1}(m - l - 1)!$ because clauses of length more than l are taken away when CNF formula $f_1^C \wedge f_2^C$ is transformed into DNF formulas $f_1^C \bar{\wedge} f_2^C$. This completes the proof of the lemma. \square

Proof of Theorem 4.1. Assume that a monotone circuit C is such that $\Pr_{v \in I(m, s_2)}[C(v) = 1] \geq h(s_2)$ and $\Pr_{u \in O(m, s_1)}[C(u) = 0] > 2/s_1^{1/3}$ hold. To prove the theorem, it suffices to show $\text{size}(C) \geq (h/2)2^{s_1^{3/4}}$. From Lemma 4.3, the approximator circuit \bar{C} satisfies $\Pr_{v \in I(m, s_2)}[C(v) \neq \bar{C}(v)] \geq h(s_2)$ or $\Pr_{u \in O(m, s_1)}[C(u) \neq \bar{C}(u)] > 1/s_1^{1/3}$. Thus, in view of Fact 4.2 and Lemmas 4.3, 4.4, and 4.5, the size of C is at least

$$(4.3) \quad \frac{1}{2} \min \left(\frac{h(s_2)m!}{(2rs_2)^{l+1}(m - l - 1)!}, \frac{m!}{s_1^{1/3}(m/s_1^{1/6})^{r+1}(m - r - 1)!} \right).$$

The coefficient $\frac{1}{2}$ here is needed to take into account the fact that circuit C is assumed to be modified so the AND and OR gates alternate along any path from an input to the output in the circuit. An elementary calculation completes the proof. \square

5. Proof of the main theorem. The goal of this section is to prove Theorem 5.1, which says that $\lfloor (1/6) \log \log m \rfloor$ NOT gates are not enough to compute the clique function feasibly. We do not intend here to optimize the constant $1/6$ in the number of NOT gates.

THEOREM 5.1. *For any sufficiently large integer m ,*

$$\text{size}_{\lfloor (1/6) \log \log m \rfloor}(\text{CLIQUE}(m, (\log m)^{3(\log m)^{1/2}})) > 2^{(1/5)(\log m)^{(\log m)^{1/2}}}.$$

Before proceeding to the proof, we describe the idea behind the proof.

Let f be $\text{CLIQUE}(m, s)$. Suppose to the contrary that a small circuit C with t NOT gates computes f . By Theorem 3.2, there are $2^{t+1} - 1 (= \alpha)$ monotone functions f_1, \dots, f_α such that each of them can be computed by a monotone circuit and that $\cup_{i \in \{1, \dots, \alpha\}} G(f_i) \supseteq G(f)$.

All proofs of lower bounds on the size of a monotone circuit computing a certain function are based on the observation that the circuit separates the minterms and maxterms of the target function [2, 5, 8, 9, 10, 13] (see Figure 5.1). Instead of focusing on the separation of the minterms and maxterms of the target function f , we consider separating pairs of vectors (w, w') in $G(f)$, that is, pairs (w, w') such that $f(w) = 0$, $f(w') = 1$ and $\text{Ham}(w, w') = 1$. Theorem 3.2 says that, if f can be computed by a small circuit C with t NOT gates, then it follows that there exists a collection of

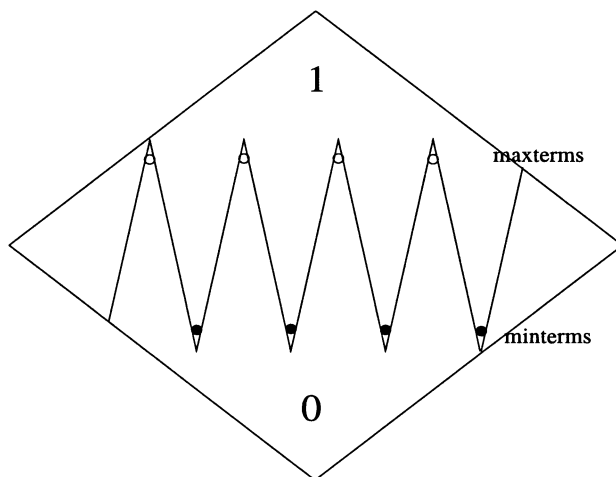


FIG. 5.1. A figure showing the minterms and maxterms of *CLIQUE* in the Boolean cube.

monotone functions f_1, \dots, f_α such that they separate $G(f)$ (i.e., $\cup_{i \in \{1, \dots, \alpha\}} G(f_i) \supseteq G(f)$) and that each of functions f_1, \dots, f_α can be computed by a small monotone circuit, where $\alpha = 2^{t+1} - 1$. In what follows we shall show that the circuit C is not small by showing that it is not the case that all of the function f_i 's satisfying the above condition can be computed by small monotone circuits. In fact we only pay attention to a subset of $G(f)$ which contains pairs that seem to be hard to separate. As the following example shows, it is crucial to decide which subset of $G(f)$ we pay attention to. As an example of a bad choice for a subset, take the subset $G'(f)$ defined as

$$G'(f) = \{(u, u^+) \mid u \in O(m, s) \text{ and } (u, u^+) \in G(f)\} \\ \cup \{(v^-, v) \mid v \in I(m, s) \text{ and } (v^-, v) \in G(f)\}.$$

Clearly, the number of 1's in u is the same for all $u \in O(m, s)$, and similarly for $v \in I(m, s)$. So if we set f_1 and f_2 to be the two threshold functions whose threshold values are the number of 1's in v and that in u^+ , then we have $G(f_1) \cup G(f_2) \supseteq G'(f)$. On the other hand, it is known that a monotone circuit of size $O(n \log n)$ can compute a threshold function on n variables for any given threshold value. So we cannot derive a contradiction by the usual approximation method argument. We will give a subset which we use as the subset of $G(f)$ consisting of edges illustrated in Figure 5.2.

Let $l_0 < l_1 < \dots < l_\alpha$ be a monotone increasing sequence of integers, where $l_0 = s$, $l_\alpha = m$, and others are chosen appropriately later. For $1 \leq i \leq \alpha$, a graph v is called *good in the i th layer* if v consists of a clique of size l_{i-1} , and contains no other edges. For $1 \leq i \leq \alpha$, a graph u is called *bad in the i th layer* if there exist l_i vertices and a partition of these l_i vertices into $s - 1$ blocks with nearly equal size such that u has an edge between any two vertices chosen from different blocks and no other edges. In other words, a bad graph in the i th layer is a $(s - 1)$ -partite complete graph on some subset of vertices of size l_i . Note that, for any good graph v in the first layer, there is an edge in $G(f)$ whose head is v because deleting an edge from v breaks the clique in v . (Recall that each vertex of the boundary graph $G(f)$ corresponds to an input graph of the clique function.) Similarly, for any bad graph u in any layer, there is an edge in $G(f)$ whose tail is u because adding an appropriate edge (between

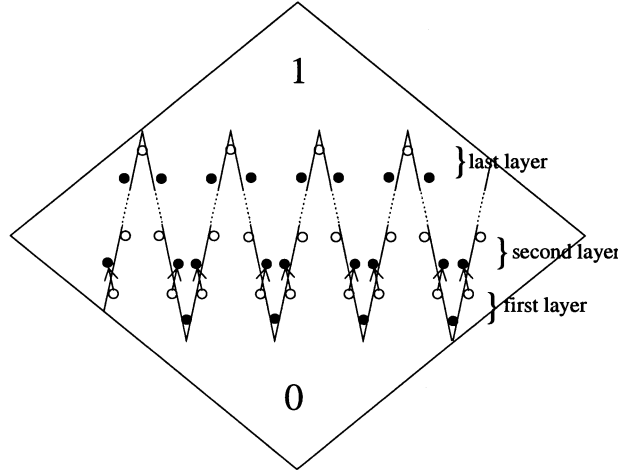


FIG. 5.2. Layered structure of the good and bad graphs: The good graphs and bad graphs are described as solid circles and open circles, respectively. A small monotone circuit cannot separate its good graphs and bad graphs in any one layer. When a bad graph u in the i th layer is connected by an arrow to a good graph v in the $(i + 1)$ th layer, they satisfy the condition $v \geq u$.

vertices in a block) to u ends up with having a clique of size s . In what follows we will prove that if monotone functions f_1, \dots, f_α separate $G(f)$ and each f_1, \dots, f_α can be computed by a small monotone circuit, then a contradiction follows. In order to prove it, because we must consider α functions which are supposed to separate $G(f)$, we must fully exploit the computational complexity of the function CLIQUE, considering α layers and focusing on the separation between good and bad graphs in each layer.

Since $\cup_{i \in \{1, \dots, \alpha\}} G(f_i) \supseteq G(f)$, there exists a function, say f_1 , in $\{f_1, \dots, f_\alpha\}$ such that $G(f_1)$ contains at least $1/\alpha$ fraction of edges of $G(f)$ ending at good graphs in the first layer, and hence f_1 outputs 1 on at least $1/\alpha$ fraction of good graphs in the first layer. Since we can use Theorem 4.1 to show that every small monotone circuit that outputs 1 for a certain fraction of good graphs in the first layer must output 1 for a large number of bad graphs in the same layer, the function f_1 takes the value 1 for a large number of bad graphs u in the first layer. By adding an edge appropriately to such a u we get a graph u^+ which contains a clique of size s . Hence there are many edges, denoted (u, u^+) , which are not included in the edges in $G(f_1)$. Since $\cup_{i \in \{1, \dots, \alpha\}} G(f_i) \supseteq G(f)$, there exists a function, say f_2 , in $\{f_2, \dots, f_\alpha\}$ such that $G(f_2)$ contains at least $1/\alpha$ fraction of such edges (u, u^+) . On the other hand, because f_2 is monotone and $f_2(u^+) = 1$, f_2 takes the value 1 on the good graph v in the second layer such that $u^+ \leq v$. Applying Theorem 4.1 again, we can conclude that f_2 outputs 1 for a large number of bad graphs in the second layer. It can be shown that f_1 also outputs 1 for such bad graphs.

By continuing the above argument, we can conclude that every function f_1, \dots, f_α outputs 1 on some bad graph u in the last layer, contradicting the fact that $\cup_{i \in \{1, \dots, \alpha\}} G(f_i) \supseteq G(f)$. This is the outline of the proof.

Proof of Theorem 5.1. Let m be a sufficiently large integer. Put $t = \lfloor (1/6) \log \log m \rfloor$, $s = (\log m)^{3(\log m)^{1/2}}$, $M = 2^{(1/5)(\log m)(\log m)^{1/2}}$, and $\alpha = 2^{t+1} - 1$. We suppose to the contrary that a circuit C with at most t NOT gates computes CLIQUE(m, s) and that

$\text{size}(C) \leq M$. From Theorem 3.2, there are monotone functions $f_1, \dots, f_\alpha \in \mathcal{M}^n$ such that $\text{size}_{\text{mon}}(f_i) \leq M$ for any $1 \leq i \leq \alpha$, and $\bigcup_{i \in \{1, \dots, \alpha\}} G(f_i) \supseteq G(\text{CLIQUE}(m, s))$.

Let $l_0 = s, l_\alpha = m$, and for every $j = 1, \dots, \alpha-1$, let $l_j = m^{1/10+(1/3)(j-1)/(\log m)^{1/6}}$. Since $l_{\alpha-1} \leq m^{1/10+(1/3)(2^{(1/6)\log \log m+1})/(\log m)^{1/6}} = m^{1/10+2/3} < m^{9/10}$, we have $l_0 < l_1 < \dots < l_\alpha$. Let V be the set of m vertices of the graph associated with CLIQUE . For $j \in \{0, \dots, \alpha\}$, let \mathcal{L}_j denote $\{L \subseteq V \mid |L| = l_j\}$ and let $\mathcal{L}_j(L)$ denote $\{L' \subseteq L \mid L' \in \mathcal{L}_j\}$. For $i \in \{1, \dots, \alpha\}$ and $L_i \in \mathcal{L}_i$, a graph v is called *good on the set L_i in the i th layer* if it consists of a clique of size l_{i-1} on some $L_{i-1} \in \mathcal{L}_{i-1}(L_i)$ (i.e., $|L_{i-1}| = l_{i-1}$ and $L_{i-1} \subseteq L_i$), and contains no other edges. For $i \in \{1, \dots, \alpha\}$ and $L_i \in \mathcal{L}_i$, a graph u is called *bad on the set L_i in the i th layer* if there is a partition of L_i into V_1, \dots, V_{s-1} such that

- (i) $|V_i| \in \{[\lfloor L_i/(s-1) \rfloor], \lceil \lceil L_i/(s-1) \rceil \rceil\}$ for $i = 1, \dots, s-1$,
- (ii) u has an edge (w, w') if and only if $w \in V_i$ and $w' \in V_j$ such that $i \neq j$, i.e., u is a complete $(s-1)$ -partite complete graph on the vertex set L_i .

Let I_{L_i} (respectively, O_{L_i}) denote the set of all good (respectively, bad) graphs on the set L_i in the i th layer. Note that a good graph in the first layer (respectively, a bad graph in the last layer) is a minterm (respectively, a maxterm) of $\text{CLIQUE}(m, s)$. We also note that there is a one-to-one correspondence between I_{L_i} and $I(l_i, l_{i-1})$, and between O_{L_i} and $O(l_i, s)$, where $I(l_i, l_{i-1})$ and $O(l_i, s)$ are defined in section 4. Hence a function in $F(l_i, s, l_{i-1})$ can be viewed as separating the graphs into two groups, I_{L_i} and O_{L_i} . Since $s^{1/3}l_{i-1} \leq l_i/200$ holds, the following corollary is straightforward from Theorem 4.1. This corollary says that a small monotone circuit cannot separate O_{L_i} and I_{L_i} for any i and for any vertex set $L_i \in \mathcal{L}_i$.

COROLLARY 5.2. *Let $i \in \{1, \dots, \alpha\}$ and $L_i \in \mathcal{L}_i$. Suppose that C is a monotone circuit and the fraction of good graphs in I_{L_i} (i.e., the set of good graphs on L_i in the i th layer) such that C outputs 1 is at least h . Then at least one of the following holds:*

- (i) *The number of gates in C is at least $(h/2)2^{s^{1/3}/4}$.*
- (ii) *The fraction of bad graphs in O_{L_i} (i.e., the set of bad graphs on L_i in the i th layer) such that C outputs 0 is at most $2/s^{1/3}$.*

Proof of Theorem 5.1 (continued). For $L \subseteq V$, let v_L denote a graph corresponding to a clique on the set L and having no other edges. Recall that $\mathcal{L}_0 = \{L \subseteq V \mid |L| = s\}$. Thus for any $L_0 \in \mathcal{L}_0$, there exists $u < v_{L_0}$ such that the edge (u, v_{L_0}) is in $G(\text{CLIQUE}(m, s))$. Hence there exists i_1 in $\{1, \dots, \alpha\}$ such that $\Pr_{L_0 \in \mathcal{L}_0}[\exists u < v_{L_0} \ (u, v_{L_0}) \in G(f_{i_1})] \geq 1/\alpha > 1/2^{t+1}$ holds, and this implies

$$(5.1) \quad \Pr_{L_0 \in \mathcal{L}_0} [f_{i_1}(v_{L_0}) = 1] \geq \frac{1}{2^{t+1}}.$$

Then we can obtain

$$(5.2) \quad \Pr_{L_1 \in \mathcal{L}_1} \left[\Pr_{v \in I_{L_1}} [f_{i_1}(v) = 1] \geq \frac{1}{2^{t+2}} \right] \geq \frac{1}{2^{t+2}}.$$

This is because from (5.1), we have

$$(5.3) \quad \begin{aligned} \sum_{L_1 \in \mathcal{L}_1} |\{v \in I_{L_1} \mid f_{i_1}(v) = 1\}| &\geq \frac{1}{2^{t+1}} \binom{m}{l_0} \binom{m-l_0}{l_1-l_0} \\ &= \frac{1}{2^{t+1}} \binom{m}{l_1} \binom{l_1}{l_0}. \end{aligned}$$

But if (5.2) does not hold, then

$$\begin{aligned} \sum_{L_1 \in \mathcal{L}_1} |\{v \in I_{L_1} \mid f_{i_1}(v) = 1\}| &< \binom{m}{l_1} \frac{1}{2^{t+2}} \binom{l_1}{l_0} + \binom{m}{l_1} \left(1 - \frac{1}{2^{t+2}}\right) \binom{l_1}{l_0} \frac{1}{2^{t+2}} \\ &< \frac{2}{2^{t+2}} \binom{m}{l_1} \binom{l_1}{l_0} = \frac{1}{2^{t+1}} \binom{m}{l_1} \binom{l_1}{l_0}, \end{aligned}$$

contradicting (5.3).

Now we call $L_1 \in \mathcal{L}_1$ *dense* if $\Pr_{v \in I_{L_1}} [f_{i_1}(v) = 1] \geq 1/2^{t+2}$ holds. Put $h = 1/2^{t+2}$. An easy calculation shows that $h \geq 1/m$. Thus by applying Claim 5.2 to every dense L_1 , we have $\text{size}_{\text{mon}}(f_{i_1}) \geq (1/2m)2^{s^{1/3}/4} = 2^{(1/4)(\log m)(\log m)^{1/2} - \log m - 1} > M$ or $\Pr_{u \in O_{L_1}} [f_{i_1}(u) = 1] \geq 1 - 2/s^{1/3} \geq 1/2$ for any dense L_1 . Since the former contradicts the assumption that $\text{size}_{\text{mon}}(f_{i_1}) \leq M$, we have $\Pr_{u \in O_{L_1}} [f_{i_1}(u) = 1] \geq 1/2$ for any dense L_1 . By (5.2), we have

$$(5.4) \quad \Pr_{L_1 \in \mathcal{L}_1} \left[\Pr_{u \in O_{L_1}} [f_{i_1}(u) = 1] \geq \frac{1}{2} \right] \geq \frac{1}{2^{t+2}}.$$

The proof will be by induction on a level of the layers. We use (5.4) as the basis of the induction, and the induction steps are as follows.

CLAIM 5.3. *Suppose $c_1 > 1$ and $c_2 > 1$. Put $c_3 = \alpha$. Let f_1, \dots, f_{c_3} be the monotone functions such that $\cup_{i \in \{1, \dots, c_3\}} G(f_i) \supseteq G(\text{CLIQUE}(m, s))$ and $\text{size}_{\text{mon}}(f_i) \leq M$ for any $1 \leq i \leq c_3$. Suppose that, for distinct indices $i_1, \dots, i_k \in \{1, \dots, c_3\}$,*

$$\Pr_{L_k \in \mathcal{L}_k} \left[\Pr_{u \in O_{L_k}} [f_{i_1}(u) = \dots = f_{i_k}(u) = 1] \geq 1/c_1 \right] \geq 1/c_2$$

holds. If $c_1 c_2 c_3 \leq s^{1/3}/8$, then there exists $i_{k+1} \in \{1, \dots, c_3\} \setminus \{i_1, \dots, i_k\}$ such that

$$\Pr_{L_{k+1} \in \mathcal{L}_{k+1}} \left[\Pr_{u \in O_{L_{k+1}}} [f_{i_1}(u) = \dots = f_{i_k}(u) = f_{i_{k+1}}(u) = 1] \geq \frac{1}{4c_1 c_2 c_3} \right] \geq \frac{1}{2c_2 c_3}.$$

For a proof of this claim, see the appendix.

Proof of Theorem 5.1 (continued). First we claim that for any $k \in \{1, \dots, \alpha\}$, there are k distinct indices $i_1, \dots, i_k \in \{1, \dots, \alpha\}$ such that

$$(5.5) \quad \Pr_{L_k \in \mathcal{L}_k} \left[\Pr_{u \in O_{L_k}} [f_{i_1}(u) = \dots = f_{i_k}(u) = 1] \geq \frac{1}{2k^2(t+2)} \right] \geq \frac{1}{2k(t+2)}$$

holds. The claim is proved by induction on k . The basis, $k = 1$, is trivial from (5.4). Now we suppose the claim holds for any $k \leq l$ and let $k = l + 1$. By the induction hypothesis, we have

$$\Pr_{L_l \in \mathcal{L}_l} \left[\Pr_{u \in O_{L_l}} [f_{i_1}(u) = \dots = f_{i_l}(u) = 1] \geq \frac{1}{2^{l^2(t+2)}} \right] \geq \frac{1}{2^{l(t+2)}}.$$

Putting $c_1 = 2^{l^2(t+2)}$, $c_2 = 2^{l(t+2)}$, and $c_3 = \alpha$, we have $4c_1 c_2 c_3 \leq 2^{2+l^2(t+2)+l(t+2)+(t+1)} \leq 2^{(l+1)^2(t+2)}$, $2c_2 c_3 \leq 2^{l+l(t+2)+t+1} = 2^{(l+1)(t+2)}$, and $c_1 c_2 c_3 \leq$

$2^{(l+1)^2(t+2)}/4 \leq 2^{(2^{t+1})^2(t+2)}/4 \leq 2^{2^{3t}}/8 \leq 2^{2^{(1/2)\log\log m}}/8 = 2^{\sqrt{\log m}}/8 < (\log m)\sqrt{\log m}/8 = s^{1/3}/8$. Thus by Claim 5.3,

$$\Pr_{L_{l+1} \in \mathcal{L}_{l+1}} \left[\Pr_{u \in O_{L_{l+1}}} [f_{i_1}(u) = \cdots = f_{i_{l+1}}(u) = 1] \geq \frac{1}{4c_1c_2c_3} \right] \geq \frac{1}{2c_2c_3}$$

holds. Therefore

$$\begin{aligned} \Pr_{L_{l+1} \in \mathcal{L}_{l+1}} \left[\Pr_{u \in O_{L_{l+1}}} [f_{i_1}(u) = \cdots = f_{i_{l+1}}(u) = 1] \geq \frac{1}{2^{(l+1)^2(t+2)}} \right] &\geq \frac{1}{2c_2c_3} \\ &\geq \frac{1}{2^{(l+1)(t+2)}}. \end{aligned}$$

This completes the induction step and hence the proof of the claim.

Recalling $\mathcal{L}_\alpha = \{V\}$ and setting k in (5.5) to α , we have $\Pr_{u \in O_V} [\forall i \in \{1, \dots, \alpha\} f_i(u) = 1] > 0$. Thus there exist $u \in O_V$ and $u^+ \in \text{CLIQUE}(m, s)^{-1}(1)$ such that $(u, u^+) \in G(\text{CLIQUE}(m, s))$ and $(u, u^+) \notin G(f_i)$ for any $i \in \{1, \dots, \alpha\}$. This implies that $\bigcup_{i \in \{1, \dots, \alpha\}} G(f_i) \not\supseteq G(\text{CLIQUE}(m, s))$, completing the proof. \square

6. Concluding remarks. There are still many interesting questions yet to be answered in the line of research pursued in the present paper. An obvious challenge is to improve the number of negation gates in the main theorem to $\omega(\log \log n)$. Another interesting problem is to show a tradeoff between the circuit size and the number of negation gates in a circuit to compute a certain monotone function. Analyzing the size complexity more carefully along the line suggested in this paper might help to explore such a tradeoff. Note that we have recently proved that such a tradeoff exists for the merging function $\text{MERGE}(n, n)$, which is a collection of monotone functions that merges two presorted binary sequence each of length n into a sorted sequence of length $2n$, by showing $\text{size}_t(\text{MERGE}(n, n)) = \Theta(n \log n / 2^t)$ for every $t = 0, \dots, \log \log n$ [3].

Finally, it should be noted that there is a large obstacle in generalizing our techniques to obtain a good lower bound for a circuit without restricting the number of negation gates. This comes from the notion of “natural proofs” introduced by Razborov and Rudich [12]. They proved that almost all known combinatorial lower bound proof techniques are “natural,” and such proofs cannot yield a good lower bound for general circuit complexity under some commonly believed cryptographic assumption. Our techniques seem to fall under the category of natural proofs although we have not tried to give a formal proof. Some radically different techniques would be needed to improve the number of negation gates in our main theorem to, say, $\log n$.

Appendix. *Proof of Claim 5.3.* Let $\mathcal{L}_k^{\text{bad}}$ denote the collection of sets $L_k \in \mathcal{L}_k$ with $\Pr_{u \in O_{L_k}} [f_{i_1}(u) = \cdots = f_{i_k}(u) = 1] \geq 1/c_1$. By the assumption of Claim 5.3, we have

$$(A.1) \quad \Pr_{L_k \in \mathcal{L}_k} [L_k \in \mathcal{L}_k^{\text{bad}}] \geq \frac{1}{c_2}.$$

Let $u \in O_{L_k}$ be such that $f_{i_1}(u) = \cdots = f_{i_k}(u) = 1$. By the definition of boundary graphs, none of $G(f_{i_1}), \dots, G(f_{i_k})$ contains an edge from u . Note that $\text{CLIQUE}(m, s)(u) = 0$. Let u^+ be a graph obtained from u by adding an arbitrary edge whose both endpoints are in L_k . Clearly, $\text{Ham}(u, u^+) = 1$, $\text{CLIQUE}(m, s)(u^+) = 1$ and $(u, u^+) \in G(\text{CLIQUE}(m, s))$. Since $u^+ \leq v_{L_k}$, we have

$$\forall L_k \in \mathcal{L}_k^{\text{bad}} \exists u \in O_{L_k} \exists u^+ \leq v_{L_k} \quad (u, u^+) \in \bigcup_{j \in \{1, \dots, c_3\} \setminus \{i_1, \dots, i_k\}} G(f_j).$$

Therefore there exists $l \in \{1, \dots, c_3\} \setminus \{i_1, \dots, i_k\}$ such that

$$\Pr_{L_k \in \mathcal{L}_k^{bad}} [\exists u \in O_{L_k} \exists u^+ \leq v_{L_k} \quad (u, u^+) \in G(f_l)] \geq \frac{1}{c_3}$$

holds. If $(u, u^+) \in G(f_l)$ for $u \in O_{L_k}$, then $f_l(u^+) = 1$, which together with $u^+ \leq v_{L_k}$ implies $f_l(v_{L_k}) = 1$ by the monotonicity of f_l . Thus we can conclude that $\Pr_{L_k \in \mathcal{L}_k} [f_l(v_{L_k}) = 1 \mid L_k \in \mathcal{L}_k^{bad}] \geq 1/c_3$. From this and (A.1), there exists $l \in \{1, \dots, c_3\} \setminus \{i_1, \dots, i_k\}$ such that

$$(A.2) \quad \Pr_{L_k \in \mathcal{L}_k} [L_k \in \mathcal{L}_k^{bad} \text{ and } f_l(v_{L_k}) = 1] \geq \frac{1}{c_2 c_3}.$$

Now we choose an index l arbitrarily that satisfies the above inequality and let $i_{k+1} = l$. Letting \mathcal{L}_k^{target} denote a collection of sets $L_k \in \mathcal{L}_k$ such that $L_k \in \mathcal{L}_k^{bad}$ and $f_{i_{k+1}}(v_{L_k}) = 1$, we have $\Pr_{L_k \in \mathcal{L}_k} [L_k \in \mathcal{L}_k^{target}] \geq 1/(c_2 c_3)$. By a similar argument to the derivation of (5.2), we have

$$(A.3) \quad \Pr_{L_{k+1} \in \mathcal{L}_{k+1}} \left[\Pr_{L_k \in \mathcal{L}_k(L_{k+1})} [L_k \in \mathcal{L}_k^{target}] \geq \frac{1}{2c_2 c_3} \right] \geq \frac{1}{2c_2 c_3}.$$

Now we call a $L_{k+1} \in \mathcal{L}_{k+1}$ *dense* if

$$(A.4) \quad \Pr_{L_k \in \mathcal{L}_k(L_{k+1})} [L_k \in \mathcal{L}_k^{target}] \geq \frac{1}{2c_2 c_3}$$

holds, and let $\mathcal{L}_{k+1}^{dense}$ denote a collection of all dense sets in \mathcal{L}_{k+1} . Note that

$$\Pr_{v \in I_{L_{k+1}}} [f_{i_{k+1}}(v) = 1] \geq 1/(2c_2 c_3)$$

for any dense $L_{k+1} \in \mathcal{L}_{k+1}^{dense}$. Put $h = 1/(2c_2 c_3) > 1/m$. Thus by applying Claim 5.2 to every dense L_{k+1} , we have $\text{size}_{mon}(f_{i_{k+1}}) > (1/2m)2^{s^{1/3}/4} > M$ or $\Pr_{u \in O_{L_{k+1}}} [f_{i_{k+1}}(u) = 0] \leq 2/s^{1/3} \leq 1/(4c_1 c_2 c_3)$ for any dense L_{k+1} . (We use the assumption $c_1 c_2 c_3 \leq s^{1/3}/8$ in Claim 5.3 here.) Since the former contradicts the assumption $\text{size}_{mon}(f_{i_{k+1}}) \leq M$, we have

$$(A.5) \quad \Pr_{u \in O_{L_{k+1}}} [f_{i_{k+1}}(u) = 0] \leq \frac{1}{4c_1 c_2 c_3},$$

for any $L_{k+1} \in \mathcal{L}_{k+1}^{dense}$. By (A.4), for any dense \mathcal{L}_{k+1} , we have

$$(A.6) \quad \Pr_{L_k \in \mathcal{L}_k(L_{k+1})} \left[\Pr_{u \in O_{L_k}} [f_{i_1}(u) = \dots = f_{i_k}(u) = 1] \geq \frac{1}{c_1} \right] \geq \frac{1}{2c_2 c_3}.$$

From the above inequality, we can get

$$(A.7) \quad \Pr_{u \in O_{L_{k+1}}} [f_{i_1}(u) = \dots = f_{i_k}(u) = 1] \geq \frac{1}{2c_1 c_2 c_3}.$$

To derive this, we consider the bipartite graph $G = (U_1, U_2, E)$ with vertex sets $U_1 = O_{L_{k+1}}$ and $U_2 = \cup_{L_k \in \mathcal{L}_k(L_{k+1})} O_{L_k}$ and the edge set

$$E = \{(u_1, u_2) \in U_1 \times U_2 \mid u_1 \geq u_2\}.$$

Let U_2^{mark} be the set of $u_2 \in U_2$ such that $f_{i_1}(u_2) = \cdots = f_{i_k}(u_2) = 1$ and let $N(U_2^{mark}) \subseteq U_1$ be the set of vertices adjacent to vertices in U_2^{mark} . By the monotonicity of f_i 's, for every $u_1 \in N(U_2^{mark})$, $f_{i_1}(u_1) = \cdots = f_{i_k}(u_1) = 1$ holds. From (A.6), we have $|U_2^{mark}| \geq |U_2|/(2c_1c_2c_3)$. Clearly every vertex in U_1 has the same degree, and similarly for U_2 . Hence we can prove that $|N(U_2^{mark})| \geq |U_1|/(2c_1c_2c_3)$, which implies (A.7).

By (A.5) and (A.7), we have

$$\Pr_{u \in O_{L_{k+1}}} [f_{i_1}(u) = \cdots = f_{i_{k+1}}(u) = 1] \geq \frac{1}{2c_1c_2c_3} - \frac{1}{4c_1c_2c_3} = \frac{1}{4c_1c_2c_3}$$

for any $L_{k+1} \in \mathcal{L}_{k+1}^{dense}$. Claim 5.3 is straightforward from this and (A.3). \square

Acknowledgment. The authors would like to thank the anonymous referees for their helpful suggestions on improving the presentation of this paper.

REFERENCES

- [1] N. ALON AND R. B. BOPPANA, *The monotone circuit complexity of Boolean functions*, *Combinatorica*, 7 (1987), pp. 1–22.
- [2] K. AMANO AND A. MARUOKA, *The potential of the approximation method*, *SIAM J. Comput.*, 33 (2004), pp. 433–447. (A preliminary version was in Proceedings of the 37th Annual Symposium on Foundations of Computer Science, Burlington, VT, 1996, pp. 431–440.)
- [3] K. AMANO, A. MARUOKA, AND J. TARUI, *On the negation-limited circuit complexity of merging*, *Discrete Appl. Math.*, 126 (2003), pp. 3–8.
- [4] R. BEALS, T. NISHINO, AND K. TANAKA, *More on the complexity of negation-limited circuits*, in Proceedings of the 27th Annual ACM Symposium on Theory of Computing, Las Vegas, NV, 1995, pp. 585–595.
- [5] C. BERG AND S. ULFBERG, *Symmetric approximation arguments for monotone lower bounds without sunflowers*, *Comput. Complexity*, 8 (1999) pp. 1–20.
- [6] M. J. FISCHER, *The complexity of negation-limited networks—A brief survey*, *Lect. Notes Comput. Sci.* 33, Springer-Verlag, Berlin, 1975, pp. 71–82.
- [7] J. HÅSTAD, *Almost optimal lower bounds for small depth circuits*, in Proceedings of the 18th Annual ACM Symposium on Theory of Computing, Berkeley, CA, 1986, pp. 6–20.
- [8] D. HARNIK AND R. RAZ, *Higher lower bounds on monotone size*, in Proceedings of the 32nd Annual ACM Symposium on Theory of Computing, Portland, OR, 2000, pp. 378–387.
- [9] S. JUKNA, *Finite limits and monotone computations: The lower bounds criterion*, in Proceedings of the 12th IEEE Conference on Computational Complexity, Ulm, Germany, 1997, pp. 302–313.
- [10] A. A. RAZBOROV, *Lower bounds on the monotone complexity of some Boolean functions*, *Dokl. Akad. Nauk SSSR*, 28 (1985), pp. 798–801.
- [11] A. A. RAZBOROV, *On the method of approximations*, in Proceedings of the 21st Annual ACM Symposium on Theory of Computing, Seattle, WA, 1989, pp. 167–176.
- [12] A. A. RAZBOROV AND S. RUDICH, *Natural proofs*, *J. Comput. System Sci.*, 55 (1997), pp. 24–35.
- [13] J. SIMON AND S. C. TSAI, *On the bottleneck counting argument*, *Theoret. Comput. Sci.*, 237 (2000), pp. 429–437.
- [14] M. SANTHA AND C. WILSON, *Limiting negations in constant depth circuits*, *SIAM J. Comput.*, 22 (1993) pp. 294–302.

BOUNDS ON THE EFFICIENCY OF GENERIC CRYPTOGRAPHIC CONSTRUCTIONS*

ROSARIO GENNARO[†], Yael GERTNER[‡], JONATHAN KATZ[§], AND LUCA TREVISAN[¶]

Abstract. A central focus of modern cryptography is the construction of efficient, high-level cryptographic tools (e.g., encryption schemes) from weaker, low-level cryptographic primitives (e.g., one-way functions). Of interest are both the *existence* of such constructions and their *efficiency*.

Here, we show essentially tight lower bounds on the best possible efficiency of any black-box construction of some fundamental cryptographic tools from the most basic and widely used cryptographic primitives. Our results hold in an extension of the model introduced by Impagliazzo and Rudich and improve and extend earlier results of Kim, Simon, and Tetali. We focus on constructions of pseudorandom generators, universal one-way hash functions, and digital signatures based on one-way permutations, as well as constructions of public- and private-key encryption schemes based on trapdoor permutations. In each case, we show that any black-box construction beating our efficiency bound would yield the unconditional existence of a one-way function and thus, in particular, prove $P \neq NP$.

Key words. lower bounds, pseudorandom generators, hash functions, digital signatures, encryption

AMS subject classifications. 94A60, 68Q17, 68P25

DOI. 10.1137/S0097539704443276

1. Introduction. A central focus of modern cryptography is the construction of high-level cryptographic protocols and tools that are both provably secure and efficient. Generally speaking, work proceeds along two lines: (1) demonstrating the *feasibility* of a particular construction, based on the weakest possible primitive; and (2) improving the *efficiency* of such constructions, either based on the weakest primitive for which a construction is known or perhaps by assuming the existence of a stronger primitive. The first of these approaches has been immensely successful; for example, the existence of one-way functions is known to be sufficient for constructing pseudorandom generators [8, 22, 27, 42], pseudorandom functions [21], universal one-way hash functions and digital signature schemes [36, 38], private-key encryption schemes and message-authentication codes [20], and commitment schemes [35]. In each of these cases one-way functions are also known to be necessary [30, 38], thus exactly characterizing the feasibility of these constructs.

*Received by the editors April 22, 2004; accepted for publication (in revised form) February 6, 2005; published electronically October 3, 2005. This work appeared in preliminary form as *Lower bounds on the efficiency of encryption and digital signature schemes*, in 35th ACM Symposium on Theory of Computing, 2003, pp. 417–425; and *Lower bounds on the efficiency of generic cryptographic constructions*, in 41st IEEE Symposium on Foundations of Computer Science, 2000, pp. 305–313.

<http://www.siam.org/journals/sicomp/35-1/44327.html>

[†]IBM T. J. Watson Research Center, Hawthorne, NY 10532 (rosario@watson.ibm.com).

[‡]Department of Psychology, University of Illinois at Urbana-Champaign, Champaign, IL 61820 (ygertner@uiuc.edu). This work was done while the author was at the Department of Computer and Information Science, University of Pennsylvania.

[§]Department of Computer Science, University of Maryland, College Park, MD 20742 (jkatz@cs.umd.edu). Portions of this research were done while the author was at DIMACS.

[¶]Computer Science Division, University of California, Berkeley, CA 94720 (luca@eecs.berkeley.edu). This work was done while the author was at Columbia University. This research was supported by NSF grant CCR-9984703.

Unfortunately, progress on the second approach—i.e., improving the efficiency of these constructions—has been much less successful. Indeed, while the constructions referred to above are all important from a theoretical point of view, their practical impact has been limited by their inefficiency. In practice, more efficient constructions based on stronger assumptions (or, even worse, heuristic solutions with no proofs of security) tend to be used. Furthermore, relying on stronger assumptions (or resorting to heuristic solutions) seems necessary to obtain improved efficiency; for each of the examples listed above, no provably secure constructions based on general assumptions are known that improve on the efficiency of the initial solutions.

This trade-off between the efficiency of a cryptographic construction and the strength of the complexity assumption on which it relies motivates the question, *How efficient can cryptographic constructions be when based on general assumptions?* We show in this paper that, in fact, the efficiency of many of the known constructions based on general assumptions *cannot be improved* without using non-black-box techniques or without finding an unconditional proof that one-way functions exist (and hence proving $P \neq NP$).

Our results hold in a generalization of the Impagliazzo–Rudich model [31], introduced by those authors in the context of proving impossibility results for the existence of certain black-box cryptographic constructions. (See section 1.3 for further discussion.) Following their work, a number of additional black-box impossibility results have appeared [13, 16, 17, 32, 40, 41]. Kim, Simon, and Tetali [33] initiated work focused on bounding the *efficiency* of black-box cryptographic constructions (rather than their *existence*), and their work provided the original inspiration for our research. We compare our results with those of Kim et al. in the following section.

1.1. Our results. Informally, we say a permutation $\pi : \{0, 1\}^n \rightarrow \{0, 1\}^n$ is *one-way with security S* if any circuit of size¹ at most S inverts π with probability less than $1/S$ (one can think of S as a slightly superpolynomial function of n but our results hold for any choice of S). In this work, we consider two types of black-box constructions, described informally now and discussed in more detail in section 1.3. Following the terminology introduced in [37], a *semi-black-box construction* (based on a one-way permutation) is an oracle procedure $P^{(\cdot)}$ such that, for any one-way permutation f given as an oracle, (1) P^f has the desired functionality and (2) P^f is “secure” (in some appropriate sense) against every efficient (oracle) adversary \mathcal{A}^f even when considering adversaries given oracle access to f . In contrast, a *weak black-box construction* is an oracle procedure $P^{(\cdot)}$ satisfying (1) as before but for which, for any one-way permutation f given as an oracle, the only guarantee is that (2) P^f is secure against efficient adversaries \mathcal{A} that are *not* given oracle access to f . Both notions preclude using the code (or circuit) of the one-way permutation f in the construction; roughly speaking (see [37] for further elaboration), semi-black-box constructions also rule out the use of the *adversary’s* code in the *security reduction* (whereas weak black-box constructions do not). Clearly, any semi-black-box construction is also a weak black-box construction and so impossibility results for the latter are stronger than impossibility results for the former.

Given these definitions, our results may be summarized informally as follows.

Pseudorandom generators (PRGs). Let U_ℓ denote the uniform distribution over ℓ -bit strings. A PRG is a deterministic, length-increasing function $G : \{0, 1\}^\ell \rightarrow \{0, 1\}^{\ell+k}$ such that $G(U_\ell)$ is computationally indistinguishable (by poly-time algo-

¹We let the *size* of a circuit refer to the number of gates the circuit has.

arithms) from $U_{\ell+k}$. The notion of a PRG was introduced by Blum and Micali [8] and Yao [42], who showed that PRGs can be constructed from any one-way permutation. This was subsequently improved by Håstad et al. [27], who show that a PRG can be constructed from any one-way function. The Blum–Micali–Yao construction, using a later improvement of Goldreich and Levin [22] (see also [18, section 2.5.3]), requires $\Theta(k/\log S)$ invocations of a one-way permutation with security S to construct a PRG stretching its input by k bits. This is the best-known efficiency for constructions based on arbitrary one-way permutations.

We show that this is essentially the best efficiency that can be obtained using black-box constructions. More formally, we show that any *weak* black-box construction of a PRG that stretches its input by k bits while making $o(k/\log S)$ invocations of a one-way permutation with security S implies the *unconditional* existence of a PRG (i.e., without any invocations of the one-way permutation). Put another way, the only way to design a more efficient construction of a PRG is to design a PRG from scratch! This would in particular imply the unconditional existence of a one-way function, as well as a proof that $P \neq NP$.

(Families of) universal one-way hash functions (UOWHFs). A UOWHF $\mathcal{H} = \{h_s\}$ is a family of length-decreasing functions (all defined over the same domain and range) such that for any input x and random choice of $h_i \in \mathcal{H}$ it is hard to find a *collision* (i.e., an $x' \neq x$ such that $h_i(x') = h_i(x)$). UOWHFs were introduced by Naor and Yung [36], who show that UOWHFs suffice to construct secure signature schemes and furthermore show how to construct the former from any one-way permutation. Rompel [38] later gave a construction of UOWHFs, and hence signature schemes, based on any one-way function. The Naor–Yung construction requires one invocation of the one-way permutation per bit of compression; that is, if $h_i : \{0, 1\}^{\ell+k} \rightarrow \{0, 1\}^\ell$ (for all $h_i \in \mathcal{H}$), then evaluating h_i requires k invocations of the one-way permutation. This can be improved easily to obtain a construction making $\Theta(k/\log S)$ invocations to compress by k bits.

We show that this, too, is essentially optimal. In particular, any *semi*-black-box construction of a UOWHF whose hash functions compress their input by k bits yet can be evaluated using $o(k/\log S)$ invocations of a one-way permutation (with security S) implies the *unconditional* existence of a UOWHF. Since the existence of UOWHFs implies the existence of one-way functions, this consequence would again imply a proof of $P \neq NP$. This improves on the work of Kim, Simon, and Tetali [33], who show a similar result but only for the case of constructions making $o(\sqrt{k}/\log S)$ invocations of a one-way permutation.

Encryption schemes. PRGs and UOWHFs may be viewed as one-party, or stand-alone, cryptographic primitives for which there is no inherent notion of interaction. We also explore the efficiency of two-party protocols, including those used in a public-key setting.

A public-key encryption scheme for m -bit messages is semantically secure [25] if for any two messages $M_0, M_1 \in \{0, 1\}^m$ the distribution over encryptions of M_0 is computationally indistinguishable from the distribution over encryptions of M_1 , even when given the public key as input. A similar definition (but with no public key) holds for the case of private-key encryption. Public-key encryption schemes constructed using the hard-core bit paradigm [8, 7, 22, 42] require $\Theta(m/\log S)$ invocations of a trapdoor permutation to encrypt an m -bit message. Similarly, private-key encryption schemes constructed using this paradigm require $\Theta(\frac{m-k}{\log S})$ invocations of a one-way permutation, where k is the length of the shared key. The same bound holds in the

private-key case even if one is willing to use the stronger assumption of a trapdoor permutation.

We show that the above constructions are essentially the best possible (at least, for the notions of security considered above). For the case of public-key encryption, we show that any *weak* black-box construction supporting encryption of m -bit messages requires $\Omega(m/\log S)$ invocations of the trapdoor permutation for the encryption algorithm alone (i.e., in addition to any invocations made during the key-generation phase). Using related techniques, we also show that any *weak* black-box construction of a private-key encryption scheme—even when based on trapdoor permutations—which securely encrypts m -bit messages using a k -bit key must query its permutation oracle $\Omega(\frac{m-k}{\log S})$ times. In each case, we show that any weak black-box construction beating our bound would imply the unconditional existence of a one-way function (from which a secure private-key encryption scheme, with no reliance on an oracle, can be derived) and hence a proof that $P \neq NP$.

Signature schemes. We say a one-time signature scheme is secure if no efficient adversary can forge a valid signature on a new message after seeing a signature on a single, random message. (We remark that it is easy to covert any such scheme to one that is secure when an adversary sees a signature on a single, *chosen* message: run two of the basic schemes in parallel to obtain keys (PK_1, SK_1) , (PK_2, SK_2) and set $PK = (PK_1, PK_2)$; to sign a message $M \in \{0, 1\}^m$ choose random $r \in \{0, 1\}^m$, sign r using SK_1 , sign $r \oplus M$ using SK_2 , and output both signatures.) Of course, lower bounds on one-time schemes immediately extend to schemes satisfying stronger definitions of security [26]. We show that in any *semi*-black-box construction of a one-time signature scheme for messages of length m based on a one-way permutation, the verification algorithm must evaluate the one-way permutation $\Omega(m/\log S)$ times. As before, any semi-black-box construction beating our bound implies the unconditional existence of a one-way function (from which a secure signature scheme requiring no oracle access can be constructed).

We observe that there exist one-time signature schemes essentially meeting our lower bound; see section 4.5 for further discussion.

1.2. Overview of our techniques. We prove our results in an extension of the model of Impagliazzo and Rudich [31, 39]. Among other things, Impagliazzo and Rudich prove that a semi-black-box construction of a secure key-exchange protocol based on a one-way permutation would inherently yield a proof that $P \neq NP$, and hence it is presumably hard to come up with such a construction. (This was later strengthened by Reingold, Trevisan, and Vadhan [37], who proved *unconditionally* that there exists no semi-black-box construction of a key-exchange protocol based on one-way permutations.) Our results are in the same vein but are in many respects even stronger. First, some of our impossibility results concern the larger class of *weak* black-box constructions. Furthermore, in all cases we show that constructions beating our bounds would imply the unconditional existence of a one-way function; this is stronger than the results of Impagliazzo–Rudich both because the existence of a one-way function is not known to be implied by $P \neq NP$ and because (in all but one case) a one-way function suffices to give an *unconditional* construction of the object under consideration. Finally, we stress that Impagliazzo and Rudich were concerned with the question of *feasibility*, while we are concerned with questions of *efficiency*.

Each of our proofs hinges on a technical result that has not been stated or proved previously: a random permutation on t -bit strings is, with high probability, one-way with security $2^{\Omega(t)}$ even against nonuniform adversaries. For the related case of

random functions, a similar result has been proved by Impagliazzo and Rudich [31] in the (much simpler) uniform case and by Impagliazzo [29] in the nonuniform case.² We also show a similar result for the case of trapdoor permutations.

Using this result, we now describe the intuition behind our lower bound using the case of PRGs as an example. Given a secure construction G of a PRG having oracle access to a one-way permutation over n -bit strings, we run G with a permutation oracle that randomly permutes its first $t = \Theta(\log S)$ bits while leaving the remaining $n - t$ bits unchanged. It follows from the technical result above that with high probability such a permutation is one-way with security S , and hence G is secure when run with a permutation chosen from this distribution. Let q be the number of queries made by G to its oracle. The key point of our proof is to notice that the answers to these q oracle queries can be simulated by a deterministic function G' *itself* (i.e., without access to any oracle) if $q \cdot t$ random bits, representing the t -prefixes of the q answers to G 's oracles queries, are included as part of the seed of G' . The distribution on the output of G' (over random choice of seed) is essentially identical to that of the output of G (over random seed, and random choice of oracle as above) and is thus indistinguishable from uniform. Finally, if q is small, then the seed-length does not grow too much and the input of G' remains shorter than its output. But this means that G' is an *unconditional* PRG which does not require any oracle access (a corollary of which is a proof that $P \neq NP$).

Additional technical work is needed to prove our bounds on UOWHFs, public-key encryption schemes, and digital signature schemes. In the latter two cases in particular, which are in a public-key setting, there is no seed as part of which to include the necessary randomness for answering oracle queries, and thus no immediate way to apply the above technique. The proofs of our lower bounds in these cases follow a slightly different approach. In the case of public-key encryption, for example, we show that a scheme making fewer than the prescribed number of queries can be used to construct (unconditionally) a secure *private-key* encryption scheme in which the key is shorter than the message. Moving from the public-key to the private-key setting circumvents the issues above and enables the necessary randomness to be included as part of the shared key without compromising the functionality or the security of the scheme. Unfortunately, our result in this case is somewhat weaker than what we obtain in all other cases; namely, we show that a *public-key* encryption scheme making few black-box oracle queries exists only if a secure *private-key* encryption scheme (or, equivalently, a one-way function) exists unconditionally. This is, of course, weaker than showing the unconditional existence of a secure public-key encryption scheme. We stress that for the case of PRGs, UOWHFs, private-key encryption schemes, and signature schemes, constructions beating our bounds imply the existence of an unconditional construction for each of these tasks.

1.3. Black-box lower bounds and impossibility results. We provide here a brief discussion regarding the notion of black-box constructions. Our presentation adapts the recent definitional work of Reingold, Trevisan, and Vadhan [37], simplifying their definitions when appropriate for the present context. The discussion here is relatively informal, and we have chosen to provide definitions specific to each primitive in the relevant subsections of section 4 rather than providing a single, generic definition as in [37].

²Although one could derive our result from Impagliazzo's result and the fact that a random function is indistinguishable from a random permutation, our proof is quite different and a bit simpler.

At a high level, a construction P of a primitive based on, say, a one-way permutation is a procedure which takes as input a permutation f and outputs a (description of a) circuit³ having some desired functionality. Informally, a construction P *uses f as a black-box* if P relies only on the input/output characteristics of the provided function f and not on any internal structure of the circuit computing f ; formally, this means the construction can be described as an oracle procedure $P^{(\cdot)}$ such that P^f itself realizes the desired functionality.

In addition to correctness (i.e., the requirement that P^f has the desired functionality for any permutation f), a construction also provides some guarantee with regard to the security of the resulting implementation P^f . We say P is a *semi-black-box construction* if the following holds:

For any hard-to-invert permutation f , the implementation P^f is secure (in some sense appropriate for the primitive being constructed) against all efficient adversaries, *even those given oracle access to f* .

In contrast, P is a *weak black-box construction* if the following relaxed definition holds:

For any hard-to-invert permutation f , the implementation P^f is secure against all efficient adversaries (who are *not* given oracle access to f).

The distinction between whether an adversary is given oracle access to f is important since the above are required to hold *even when f is not efficiently computable* (and so the only way for an efficient adversary to evaluate f , in general, may be via oracle access to f). We hasten to point out, however, that even weak black-box constructions suffice to give implementations with meaningful security guarantees in the real world: in this case, f *will* be efficiently computable, and furthermore an explicit circuit for f will be known; hence, it is irrelevant whether an adversary is given oracle access to f .

Clearly, any semi-black-box construction is also a weak black-box construction, and hence impossibility results for the latter are stronger. Indeed, Reingold, Trevisan, and Vadhan [37] show that (1) a semi-black-box construction of key exchange from one-way functions is unconditionally impossible, yet (2) (informally) if the statement “one-way functions imply key exchange” is true, then there does exist a weak black-box construction of key exchange (for a single-bit key) from one-way functions. Roughly speaking, a difference between semi-black-box and weak black-box constructions is with regard to whether the circuits of *adversaries* attacking P^f can be used in the security *reduction* (i.e., the proof that P is secure). In more detail, typical security proofs for a construction P proceed by showing via *reduction* how any efficient adversary \mathcal{A}_P breaking the security of P^f can be used to construct an efficient adversary \mathcal{A}_f inverting f . Since the latter is impossible by assumption on f , this implies that no \mathcal{A}_P with the claimed properties exists. If the reduction relies only on the input/output characteristics of \mathcal{A}_P , we refer to this as a reduction which *uses the adversary as a black-box*. In contrast, a reduction which relies on knowledge of the circuit for \mathcal{A}_P is said to *make non-black-box use of the adversary*. A security reduction for a *weak* black-box construction may potentially make non-black-box use of the adversary. As argued in [37], however, security reductions for *semi*-black-box constructions are essentially restricted to using the adversary as a black-box (see [37] for further discussion).

Non-black-box constructions. Black-box constructions form an important

³For simplicity, the present discussion is phrased in terms of circuits rather than Turing machines, although similar definitions could be made in the latter case as well.

subclass, since most cryptographic constructions are black-box (indeed, most known constructions are *fully black-box* [37], a notion which is even more restrictive than semi-black-box). We stress, however, that a number of non-black-box cryptographic constructions are known.⁴ The examples of which we are aware occur in two ways: due to the use of generic zero-knowledge proofs (of knowledge) [23, 12, 4] or due to the use of generic protocols for secure computation [43, 24]. As an illustrative example, let L_f denote the image of a function f ; i.e., $L_f \stackrel{\text{def}}{=} \{y | y = f(x)\}$. A cryptographic protocol which utilizes a zero-knowledge proof that $y \in L_f$ (for f a one-way function, say) requires the parties to agree on a circuit computing f and is thus inherently non-black-box.⁵ Examples of non-black-box constructions where zero-knowledge proofs of this sort are used include a construction of an identification protocol based on one-way functions [12], a signature scheme based on noninteractive zero-knowledge [5], and *all* known constructions of chosen-ciphertext-secure encryption schemes from trapdoor permutations (e.g., [11]). Furthermore, protocols for distributed computation (without honest majority) tolerating computationally bounded, malicious adversaries [24, 43] are *themselves* non-black-box.⁶ (This is in contrast to the case of zero-knowledge proofs; cf. footnote 5.)

Knowledge of the circuit computing f is also necessary if one wants to evaluate f in a secure, distributed fashion (e.g., if two parties with respective inputs x_1, x_2 want to evaluate $y = f(x_1 \oplus x_2)$ without revealing any more information about their inputs than what is revealed by y itself). Thus, a generic construction of a threshold cryptosystem [10] based on a family F of trapdoor permutations (in which the parties share the trapdoor for inverting a single member of this family) would inherently make non-black-box use of the underlying circuit(s) for F . Another example is a result of Beaver [3], which makes non-black-box use of a one-way function f to extend “few” oblivious transfers into “many.”

Given the above, a black-box impossibility result cannot be said to rule out the feasibility of a particular construction. Yet, it is unclear how non-black-box techniques can help outside the domains mentioned above (i.e., generic use of zero-knowledge proofs or secure computation). Furthermore, a black-box impossibility result is useful insofar as it indicates the type of techniques that will be necessary to achieve a desired result or, conversely, the type of techniques that are ruled out. Finally, it is fair to say that non-black-box constructions are much less efficient than black-box ones (this is certainly the case for all the examples given above, and we are aware of no exceptions), and thus a black-box impossibility result does seem to rule out constructions likely to be *practical*.

1.4. Future work and open problems. This work suggests a number of intriguing research directions. The results given here suggest that assuming only the existence of one-way permutations (or, in some cases, even trapdoor permutations) may be too weak of a computational hypothesis to obtain *efficient* cryptographic con-

⁴We focus here on constructions making non-black-box use of an underlying function f , rather than on constructions whose security analysis makes non-black-box use of the adversary (as in [1, 2]).

⁵Note that constructions of zero-knowledge proofs for NP (e.g., [23]) are *themselves* black-box in their usage of primitives such as one-way functions. The issue is that a proof for the language of interest—e.g., L_f in the example in the text—cannot be given unless a (poly-size) circuit computing f is available.

⁶For example, although the well-known protocol for oblivious transfer secure against *semi-honest* adversaries [19, section 7.3.2] makes black-box use of trapdoor permutations, adapting the protocol (using zero-knowledge proofs) to ensure security against *malicious* adversaries involves *non-black-box* use of the circuit for the trapdoor permutation.

structions. Thus, stronger assumptions may be needed to build practical schemes. It is important to explore the minimal such assumptions necessary to achieve greater efficiency, as well as to bound the maximum achievable efficiency even when such stronger assumptions are made. For example, what additional efficiency is possible if *homomorphic* one-way permutations (i.e., permutations over a group G satisfying $f(ab) = f(a)f(b)$ for all $a, b \in G$) are assumed?

In a related vein, it will be interesting to explore more efficient constructions based on specific number-theoretic assumptions. As will be evident from our proof techniques, the efficiency limitations of constructions based on arbitrary (trapdoor) one-way permutations stem from the fact that a one-way permutation may have security S even if it has only $\Theta(\log S)$ hard-core bits. (Actually, we use pathological functions of this form to prove our lower bounds.) But specific one-way permutations and trapdoor permutations with $\Theta(n)$ hard-core bits are known under suitable number-theoretic assumptions (e.g., [28, 9]). Given such functions, we know how to construct PRGs and semantically secure private- and public-key encryption schemes with improved efficiency. It remains open, however, whether such functions can also be used to improve the efficiency of digital signature schemes or (say) public-key encryption schemes achieving chosen-ciphertext security.

The present work also leaves some more concrete open questions. First, can bounds on the efficiency of other cryptographic constructions (e.g., commitment schemes) also be given? Additionally, our lower bounds essentially match known upper bounds only for schemes achieving relatively weak notions of security, namely, semantic security for encryption of a *single* message in the case of encryption and one-time security for the case of signatures. What can be said about schemes achieving stronger notions of security? Examples of interest include private-key encryption schemes secure when polynomially many messages are encrypted (this seems related to the efficiency of pseudorandom *functions*, for which a gap remains between known upper and lower bounds), public-key encryption schemes satisfying various flavors of nonmalleability/security against chosen-ciphertext attacks, and signature schemes secure when polynomially many messages are signed.

Finally, our bound for signatures pertains to the efficiency of signature verification. It would be nice to have corresponding bounds for the efficiency of key-generation/signing.

2. Definitions and preliminaries. In this section, we review some notation and definitions for the various standard cryptographic primitives considered in this work. Appropriate definitions of black-box constructions are deferred to the relevant sections containing our lower bounds.

In what follows, we consider a number of definitions of (S, ε) -security having the form “no circuit of size S can have advantage better than ε ,” where advantage is defined in some appropriate way. In each of the cases considered here, the existence of an (S, ε) -secure scheme for any $\varepsilon < 1$ implies the existence of an (S', ε') -secure scheme for an arbitrarily small $\varepsilon' > 0$. For this reason, in all our lower bounds we content ourselves with showing the existence of an (S, ε) -secure construction for an arbitrary $\varepsilon < 1$.

All our results are stated and proved in the nonuniform setting for simplicity only; we stress that they extend immediately to the uniform setting as well. Indeed, it is for this reason that we claim that constructions beating our efficiency bounds imply the existence of one-way functions and $P \neq NP$. For example, Theorem 4.2 states only that the existence of a certain PRG construction $G^{(\cdot)} : \{0, 1\}^\ell \rightarrow \{0, 1\}^{\ell+k}$

based on a permutation $\pi : \{0, 1\}^n \rightarrow \{0, 1\}^n$ implies the unconditional existence of a PRG $G' : \{0, 1\}^{\ell'} \rightarrow \{0, 1\}^{\ell'+k}$. Essentially the same proof, however, also shows that a certain construction of a PRG family $\{G_i^{(\cdot)} : \{0, 1\}^{\ell(i)} \rightarrow \{0, 1\}^{\ell(i)+k(i)}\}$ based on a one-way permutation family $\{\pi_i : \{0, 1\}^{n(i)} \rightarrow \{0, 1\}^{n(i)}\}$, where $G_i^{(\cdot)}$ can be evaluated by a uniform algorithm in polynomial time (in i), implies the unconditional existence of a PRG family $\{G'_i : \{0, 1\}^{\ell'(i)} \rightarrow \{0, 1\}^{\ell'(i)+k(i)}\}$, where again each G'_i can be evaluated by a uniform algorithm in polynomial time.

2.1. One-way permutations and trapdoor permutations. We say that a function $\pi : \{0, 1\}^n \rightarrow \{0, 1\}^n$ is (S, ε) -one way if for every circuit A of size $\leq S$ we have $\Pr_x[A(f(x)) \in f^{-1}(f(x))] \leq \varepsilon$. When f is given as an oracle, we provide A with access to f and write this as A^f . To reduce the number of parameters, we will call a function S -hard if it is $(S, 1/S)$ -one way.

Let Π_t denote the set of all permutations over $\{0, 1\}^t$. In section 3.1 we prove the following theorem.

THEOREM 2.1. *For all sufficiently large t , a random $\pi \in \Pi_t$ is $2^{t/5}$ -hard with probability at least $1 - 2^{-2^{t/2}}$.*

For $t \leq n$, let $\Pi_{t,n}$ denote the subset of Π_n such that $\pi \in \Pi_{t,n}$ iff $\pi(a, b) = (\hat{\pi}(a), b)$ for some $\hat{\pi} \in \Pi_t$ (that is, π permutes the first t bits of its input, while leaving the remaining $n - t$ bits fixed). An immediate corollary of the above theorem is that if $t = 5 \log S$, then for any $n \geq t$ a random $\pi \in \Pi_{t,n}$ is S -hard with very high probability.

COROLLARY 2.2. *For all sufficiently large t and any $n \geq t$, a random $\pi \in \Pi_{t,n}$ is $2^{t/5}$ -hard with probability greater than $1 - 2^{-2^{t/2}}$.*

Our model for (one-way) trapdoor permutations is somewhat more involved. We represent a family of trapdoor permutations as a tripartite oracle $\tau = (G, F, F^{-1})$. Informally, G corresponds to the key generation oracle which when queried on a string td (intended as a trapdoor) produces the corresponding public key k . The oracle F is the actual trapdoor permutation, which will be queried on key k and an input x . The oracle F^{-1} allows inversion of F ; i.e., if $G(td) = k$ and $F(k, x) = y$, then $F^{-1}(td, y) = x$.

More formally, consider the class $T_n = \{\tau \mid \tau = (G, F, F^{-1})\}$ where

- $G \in \Pi_n$ is a permutation over $\{0, 1\}^n$ (having G map trapdoors to keys rather than having G map seeds to a (trapdoor, key) pair does not affect our results);
- $F : \{0, 1\}^n \times \{0, 1\}^n \rightarrow \{0, 1\}^n$ is an oracle such that, for each $k \in \{0, 1\}^n$, $F(k, \cdot)$ is a permutation on $\{0, 1\}^n$;
- $F^{-1} : \{0, 1\}^n \times \{0, 1\}^n \rightarrow \{0, 1\}^n$ is an oracle defined as follows: $F^{-1}(td, y)$ returns the unique x such that $G(td) = k$ and $F(k, x) = y$.

A uniformly random $\tau = (G, F, F^{-1}) \in T_n$ is chosen in the natural way: G is chosen at random from Π_n and, for each $k \in \{0, 1\}^n$, the permutation $F(k, \cdot)$ is chosen independently at random from Π_n . We say that trapdoor permutation family $\tau = (G, F, F^{-1})$ is (S, ε) -trapdoor one way if for every circuit A of size $\leq S$ we have

$$\Pr_{x, td}[k := G(td) : A^\tau(k, F(k, x)) = x] \leq \varepsilon.$$

We say that τ is S -trapdoor one way if it is $(S, 1/S)$ -trapdoor one way. When clear from the context, we will also say that τ is S -hard. Although technically one must always speak of families of trapdoor permutations, we will often abuse terminology and simply refer to a $\tau \in T_n$ as a trapdoor permutation.

In section 3.2, we prove the following analogue of Theorem 2.1 for trapdoor permutations.

THEOREM 2.3. *For all sufficiently large t , a random $\tau \in T_t$ is $2^{t/5}$ -hard with probability greater than $1 - 2^{-2^{t/2}}$.*

For $t \leq n$, we let $T_{t,n} \subseteq T_n$ be defined as follows: $\tau = (G, F, F^{-1}) \in T_{t,n}$ iff the following hold:

- $G \in \Pi_{t,n}$, and thus $G(td_a, td_b) = (\hat{G}(td_a), td_b)$ for some $\hat{G} \in \Pi_t$.
- $F((k_a, k_b), (x_a, x_b)) = (\hat{F}(k_a, x_a), x_b)$, where $\hat{F}(k_a, \cdot) \in \Pi_t$. Equivalently, $F(k, \cdot) \in \Pi_{t,n}$ and furthermore this permutation is determined by the first t bits of k .
- As before, $F^{-1}(td, y)$ returns the unique x s.t. $G(td) = k$ and $F(k, x) = y$.

An immediate corollary of Theorem 2.3 is that if $t = 5 \log S$, then for any $n \geq t$ a random $\tau \in T_{t,n}$ is S -hard with very high probability.

COROLLARY 2.4. *For all sufficiently large t and any $n \geq t$, a random $\tau \in T_{t,n}$ is $2^{t/5}$ -hard with probability greater than $1 - 2^{-2^{t/2}}$.*

2.2. Pseudorandom generators. We say that two distributions X, Y are (S, ε) -indistinguishable if for every distinguishing circuit Dist of size at most S we have

$$\left| \Pr_{x \in X} [\text{Dist}(x) = 1] - \Pr_{y \in Y} [\text{Dist}(y) = 1] \right| \leq \varepsilon.$$

We also write this as $X \stackrel{(S, \varepsilon)}{\approx} Y$. We say that a function $G : \{0, 1\}^\ell \rightarrow \{0, 1\}^{\ell+k}$ is an (S_g, ε) -secure PRG if $G(U_\ell)$ is (S_g, ε) -indistinguishable from $U_{\ell+k}$, where U_n denotes the uniform distribution over $\{0, 1\}^n$.

2.3. Universal one-way hash functions. As discussed in the introduction, a family of UOWHFs is a family \mathcal{H} of length-decreasing functions such that, for a random function $h \in \mathcal{H}$ and a random point x in the domain, it is hard (given h, x) to find $x' \neq x$ such that $h(x') = h(x)$. More formally, a family $\mathcal{H} = \{h_s : \{0, 1\}^{\ell+k} \rightarrow \{0, 1\}^\ell\}_{s \in \{0, 1\}^r}$ of functions is an (S, ε) -UOWHF if for every circuit A of size at most S we have

$$\Pr_{s,x} [A(s, x, h_s(x)) = x' : x' \neq x \wedge h_s(x') = h_s(x)] \leq \varepsilon.$$

We will represent such a family as a single function $H : \{0, 1\}^r \times \{0, 1\}^{\ell+k} \rightarrow \{0, 1\}^\ell$, where $H(s, x) = h_s(x)$.

2.4. Public- and private-key encryption.

2.4.1. Public-key encryption. A public-key encryption scheme for m -bit messages is a tuple of algorithms $\mathcal{PKC} = (\text{Gen}, \text{Enc}, \text{Dec})$ having the following functionality:

- The *key generation algorithm* Gen is a probabilistic algorithm which generates a key pair (pk, sk) . We say pk is the public key and sk is the secret key.
- The *encryption algorithm* Enc is a probabilistic algorithm which, on input a public key pk and a message $M \in \{0, 1\}^m$, outputs a ciphertext C .
- The *decryption algorithm* Dec is a deterministic algorithm which, on input a secret key sk and a ciphertext C , outputs a message $M \in \{0, 1\}^m$ or \perp .

We also require perfect correctness; that is, for all (pk, sk) output by Gen , all $M \in \{0, 1\}^m$, and all C output by $\text{Enc}(pk, M)$, we have $\text{Dec}(sk, C) = M$. (Our results can be modified appropriately for the case of decryption schemes with error; see the remark following Lemma 4.6.)

For $M \in \{0, 1\}^m$, let $\mathcal{PK}\mathcal{E}(M)$ denote the distribution on the view of an adversary eavesdropping on the encryption of message M ; i.e.,

$$\mathcal{PK}\mathcal{E}(M) \stackrel{\text{def}}{=} \{(pk, sk) \leftarrow \text{Gen}; C \leftarrow \text{Enc}(pk, M) : (pk, C)\}.$$

We say that $\mathcal{PK}\mathcal{E}$ is (S_e, ε) -secure if for all $M_0, M_1 \in \{0, 1\}^m$ we have

$$\mathcal{PK}\mathcal{E}(M_0) \stackrel{(S_e, \varepsilon)}{\approx} \mathcal{PK}\mathcal{E}(M_1).$$

This corresponds to a definition of indistinguishability, or semantic security [25].

The above can be extended in the natural way to allow for interactive encryption schemes. However, since we do not explicitly consider interactive public-key encryption in this work and since a formal definition of interactive encryption in the private-key setting is given below, we omit the details.

2.4.2. Private-key encryption. The model for private-key encryption is an appropriate modification of the above. A private-key encryption scheme for m -bit messages using k -bit keys is a pair of algorithms $\mathcal{SK}\mathcal{E} = (\text{Enc}, \text{Dec})$, where

- the encryption algorithm Enc is a probabilistic algorithm which, on input a key $sk \in \{0, 1\}^k$ and a message $M \in \{0, 1\}^m$, outputs a ciphertext C ;
- the decryption algorithm Dec is a deterministic algorithm which, on input a key $sk \in \{0, 1\}^k$ and a ciphertext C , outputs either a message $M \in \{0, 1\}^m$ or \perp .

As in the public-key case, we require perfect correctness: i.e., for all $sk \in \{0, 1\}^k$, all $M \in \{0, 1\}^m$, and all C output by $\text{Enc}(sk, M)$, we have $\text{Dec}(sk, C) = M$. (Our results can be modified appropriately for the case of decryption schemes with error; see the remark following Lemma 4.6.)

For $M \in \{0, 1\}^m$, denote by $\mathcal{SK}\mathcal{E}(M)$ the probability distribution over the view of an adversary eavesdropping on the encryption of message M (where the shared key sk is chosen uniformly at random); i.e.,

$$\mathcal{SK}\mathcal{E}(M) \stackrel{\text{def}}{=} \{sk \leftarrow \{0, 1\}^k; C \leftarrow \text{Enc}(sk, M) : C\}.$$

We say that $\mathcal{SK}\mathcal{E}$ is (S_e, ε) -secure if for all $M_0, M_1 \in \{0, 1\}^m$ we have

$$\mathcal{SK}\mathcal{E}(M_0) \stackrel{(S_e, \varepsilon)}{\approx} \mathcal{SK}\mathcal{E}(M_1).$$

The above can be extended to allow for interactive encryption in the natural way. In this case, Enc and Dec represent interactive Turing machines operating in a sequence of rounds. For notational convenience, we let $T \leftarrow \widehat{\text{Enc}}(sk, M)$ denote the experiment in which random coins ω_1, ω_2 are chosen for Enc and Dec , respectively, and T is the transcript resulting from the interaction of $\text{Enc}(sk, M; \omega_1)$ with $\text{Dec}(sk; \omega_2)$. We also let $M' \leftarrow \widehat{\text{Dec}}(sk, M)$ denote the final output of Dec at the conclusion of the above experiment. Perfect correctness requires that $\widehat{\text{Dec}}(sk, M) = M$ with probability 1. In the interactive setting, $\mathcal{SK}\mathcal{E}(M)$ denotes the distribution

$$\mathcal{SK}\mathcal{E}(M) \stackrel{\text{def}}{=} \left\{ sk \leftarrow \{0, 1\}^k; T \leftarrow \widehat{\text{Enc}}(sk, M) : T \right\};$$

definitions for (S_e, ε) -security follow in the obvious way.

2.5. Signature schemes. A signature scheme for m -bit messages is a tuple of algorithms $\text{SIG} = (\text{Gen}, \text{Sign}, \text{Vrfy})$ having the following functionality:

- The *key generation algorithm* Gen is a probabilistic algorithm which generates a key pair (PK, SK) , where PK is the *public key* and SK is the *secret key*.
- The *signing algorithm* Sign is a probabilistic algorithm which, on input SK and a message $M \in \{0, 1\}^m$, outputs a signature σ .
- The *verification algorithm* Vrfy is a deterministic algorithm which, on input PK , a message M , and a signature σ , outputs a single bit.

We also require that for all (PK, SK) output by Gen , all $M \in \{0, 1\}^m$, and all σ output by $\text{Sign}(SK, M)$ we have $\text{Vrfy}(PK, M, \sigma) = 1$.

Our definition of security for signature schemes is extremely weak: we require security against existential forgery only for an adversary who gets a signature on a single, random message (i.e., we consider a *one-time signature scheme* secure under *random-message attack*). Our lower bounds apply even to weakly secure schemes of this type. Since any signature scheme secure against adaptive chosen-message attack (cf. [26]) trivially achieves this “weak” level of security, our results immediately imply a bound for the more general case. Formally, signature scheme SIG is (S_Σ, ε) -secure if for all circuits A of size at most S_Σ we have

$$\Pr \left[\begin{array}{l} (PK, SK) \leftarrow \text{Gen}; M \leftarrow \{0, 1\}^m; \\ \sigma \leftarrow \text{Sign}(SK, M); (M', \sigma') := A(PK, M, \sigma); \\ \text{Vrfy}(PK, M', \sigma') = 1 \wedge M' \neq M \end{array} \right] \leq \varepsilon.$$

3. Hardness of random (trapdoor) permutations. In this section, we state and prove two key technical theorems which show that a random permutation $\pi \in \Pi_t$ and a random trapdoor permutation $\tau \in T_t$ are exponentially hard with all but negligible probability for t large enough. Before doing so, we first state the following bound on the number of oracle circuits of a given size S .

LEMMA 3.1. *The number of circuits of size S having input/output length n and oracle access to a function $f : \{0, 1\}^n \rightarrow \{0, 1\}^n$ is at most $2^{2S+(S+1)n(\log(Sn+n)+1)}$.*

Proof. A circuit of the form considered here consists of three types of gates: and gates, or gates, and oracle gates; the first two have in-degree 2 and out-degree 1, while oracle gates have in-degree and out-degree n . A circuit may be specified by listing for each gate its type and, for each of this gate’s at most n input wires, the source of the wire (which may be the output wire of another gate or one of the input wires of the circuit) and whether its value is complemented. Finally, for each of the n output wires of the circuit one must also specify the source of this wire and whether its value is complemented. The information associated with each gate can be specified using at most $2 + n(\log(Sn + n) + 1)$ bits (per gate), while the information associated with each output wire can be specified using at most an additional $\log(Sn + n) + 1$ bits. The lemma follows. \square

3.1. Hardness of random permutations. We now prove Theorem 2.1, restated for convenience.

THEOREM 3.2. *For all sufficiently large t , a random $\pi \in \Pi_t$ is $2^{t/5}$ -hard with probability at least $1 - 2^{-2^{t/2}}$.*

Proof. We begin by showing that given any (π, A) such that A inverts π with high probability, the permutation π has a short description (given A).

CLAIM. *Let A be a circuit that makes q queries to a permutation $\pi : \{0, 1\}^t \rightarrow \{0, 1\}^t$ and for which $\Pr_y[A^\pi(y) = \pi^{-1}(y)] \geq \varepsilon$. Then π can be described using at*

most

$$2 \log \binom{2^t}{a} + \log ((2^t - a)!)$$

bits (given A), where $a = \varepsilon 2^t / (q + 1)$.

Proof of claim. Let $N = 2^t$. Consider the set I of at least εN points on which A is able to invert π , after making q queries to π . We argue that there exists a set $Y \subseteq I$ such that $|Y| \geq a$ and such that the value of π^{-1} is completely determined by the circuit A , the sets Y and $X \stackrel{\text{def}}{=} \pi^{-1}(Y)$, and the value of π^{-1} on all points in $\{0, 1\}^t \setminus Y$.

Define Y via the following process. Initially Y is empty, and all elements in I are candidates for inclusion in Y . Take the lexicographically first element y from I , and place it in Y . Next, simulate the computation of $A^\pi(y)$ and let x_1, \dots, x_q be the queries made by A to π (we assume without loss of generality that they are different), and let y_1, \dots, y_q be the corresponding answers (i.e., $y_i = \pi(x_i)$). If $y \notin \{y_i\}_{i=1}^q$, then remove y_1, \dots, y_q from I . If $y = y_i$ for some i , then remove y_1, \dots, y_{i-1} from I . Then take the lexicographically smallest of the remaining elements of I , put it into Y , etc. At any step of the construction, one element is added to Y and at most q are removed from I . Since I initially contains at least εN elements, in the end we have $|Y| \geq \lceil \varepsilon N / q \rceil > \varepsilon N / (q + 1)$.

We claim that given descriptions of the sets Y and $X = \pi^{-1}(Y)$, the values of π on $\{0, 1\}^t \setminus X$, and the circuit A , it is possible to invert (or, equivalently, compute) π everywhere. For $y \notin Y$, the value of $\pi^{-1}(y)$ is explicitly given. The values of π^{-1} on Y can be reconstructed sequentially for all $y \in Y$, taken in lexicographic order, as follows. Simulate the computation of $A^\pi(y)$. By construction of Y , during its computation $A^\pi(y)$ will query π either on points not in X , on points $x \in X$ for which $\pi(x) <_{lex} y$, or on the point $x \in X$ for which $\pi(x) = y$. In the first two cases, we have enough information to continue the simulation. In the last case, the query itself gives the desired answer $\pi^{-1}(y)$. In all possible cases, we have enough information to reconstruct $\pi^{-1}(y)$.

Describing Y , X , and the values of π on $\{0, 1\}^t \setminus X$ requires

$$2 \log \binom{N}{|Y|} + \log ((N - |Y|)!)$$

bits, which is at most the number of bits claimed.

Given the above claim, we may now easily prove the theorem. Let A be an oracle circuit of size at most $S = 2^{t/5}$. Note that A will make at most $q = 2^{t/5}$ queries to π . Let $N = 2^t$. From the claim, we see that the fraction of permutations $\pi \in \Pi_t$ such that

$$(1) \quad \Pr_x [A^\pi(\pi(x)) = x] \geq 2^{-t/5}$$

is at most

$$\frac{\binom{N}{a}^2 (N - a)!}{N!} = \frac{\binom{N}{a}}{a!},$$

where $a = 2^{-t/5} 2^t / (2^{t/5} + 1) \geq N^{3/5} / 2$. Using the inequalities $a! \geq (a/e)^a$ and $\binom{N}{a} \leq (eN/a)^a$, we may derive the following upper bound on the above expression:

$$\frac{\binom{N}{a}}{a!} \leq \left(\frac{e^2 N}{a^2} \right)^a < \left(\frac{4e^2}{N^{1/5}} \right)^a < 2^{-a} < 2^{-N^{3/5}/2}$$

for all sufficiently large N .

By Lemma 3.1 there are at most $2^{2S+(S+1)t(\log(S+t)+1)} = 2^{\tilde{O}(N^{1/5})}$ circuits of size S (where the \tilde{O} -notation suppresses polylogarithmic factors). A union bound thus shows that the probability over a random choice of $\pi \in \Pi_t$ that there *exists* a circuit of size S for which (1) holds is at most

$$2^{\tilde{O}(N^{1/5})} \cdot 2^{-N^{3/5}/2} < 2^{-N^{1/2}}$$

for all sufficiently large N . \square

3.2. Hardness of random trapdoor permutations. We now prove an analogue of the above theorem for trapdoor permutations. (This is Theorem 2.3, restated for convenience.)

THEOREM 3.3. *For all sufficiently large t , a random $\tau \in T_t$ is $2^{t/5}$ -hard with probability greater than $1 - 2^{-2^{t/2}}$.*

Proof. The proof is substantially similar to the proof of Theorem 2.1. We first prove the following claim.

CLAIM. *Let A be a circuit making q queries to a trapdoor permutation $\tau \in T_t$ and for which $\Pr_{k,y}[A^\tau(k,y) = x \wedge F(k,x) = y] \geq \varepsilon$. Then τ can be described using at most*

$$1 + 2^t \log(2^t!) + t + 2 \log \binom{2^t}{a} + \log((2^t - a)!)$$

bits (given A), where $a = \varepsilon 2^t / (2q + 1)$.

Proof of claim. Let $N = 2^t$, and let $Q(k, y)$ denote the event that $A^\tau(k, y)$ queries either $G(td)$ or $F^{-1}(td, y')$, where y' is arbitrary and $G(td) = k$. Also, we say A^τ inverts (k, y) if $A^\tau(k, y) = x$ such that $F(k, x) = y$. There are two possibilities: either

$$(2) \quad \Pr_{k,y}[A^\tau \text{ inverts } (k, y) \wedge Q(k, y)] \geq \varepsilon/2$$

or

$$(3) \quad \Pr_{k,y}[A^\tau \text{ inverts } (k, y) \wedge \overline{Q(k, y)}] \geq \varepsilon/2.$$

Consider the first case. Here, there certainly exists a \hat{y} for which it is the case that $\Pr_k[A^\tau \text{ inverts } (k, \hat{y}) \wedge Q(k, \hat{y})] \geq \varepsilon/2$. Proceeding as in the proof of Theorem 2.1, we will specify G using a small number of bits. Let I be the set of at least $\varepsilon N/2$ points (k, \hat{y}) on which A^τ inverts (k, \hat{y}) and $Q(k, \hat{y})$ occurs. Define a set $K \subseteq I$ via the following process. Initially K is empty, and all elements in I are candidates for inclusion in K . Take the lexicographically first element (k, \hat{y}) from I and place it in K . Next, simulate the computation of $A^\tau(k, \hat{y})$. Since $Q(k, \hat{y})$ occurs, we know that there is a query i of the form $G(td)$ or $F^{-1}(td, y')$, where $G(td) = k$. Looking at each of the preceding $i - 1$ queries, for each query of the form $G(td')$ with answer k' remove (k', \hat{y}) from I . For each query of the form $F^{-1}(td', y')$ let $G(td') = k'$ and remove (k', \hat{y}) from I . Then take the lexicographically smallest of the remaining elements of I , put it into K , etc. At any step of the construction, one element is added to K and at most q are removed from I . Since I initially contains at least $\varepsilon N/2$ elements, in the end we have $|K| \geq \lceil \varepsilon N/2q \rceil > \varepsilon N/(2q + 1)$.

Exactly as in the proof of Theorem 2.1 (and so we omit the details), the permutation G is completely specified given A , \hat{y} , descriptions of K and $G^{-1}(K)$, the

values of G^{-1} on $\{0, 1\}^t \setminus K$, and the values of F on all points. This requires $2 \log \binom{N}{|K|} + \log((N - |K|)!)$ bits plus an additional $2^t \log(2^t!)$ bits to specify F . Using an additional bit to specify that we are in the first case, we can completely specify τ in at most the number of bits claimed.

Consider next the second case (i.e., when (3) holds). Here, there must exist a \hat{k} for which $\Pr_y[A^\tau \text{ inverts } (\hat{k}, y) \wedge Q(\hat{k}, y)] \geq \varepsilon/2$. Now, we specify $F(\hat{k}, \cdot)$ using a small number of bits. Let I be the set of at least $\varepsilon N/2$ points (\hat{k}, y) on which A^τ inverts (\hat{k}, y) and $Q(\hat{k}, y)$ does not occur. Define a set $Y \subseteq I$ via the following process. Initially Y is empty, and all elements in I are candidates for inclusion in Y . Take the lexicographically first element (\hat{k}, y) from I and place it in Y . Next, simulate the computation of $A^\tau(\hat{k}, y)$. Consider the $\ell \leq q$ queries made by A of the form $\{F(\hat{k}, x_i)\}$ with corresponding answers $\{y_i\}$. If $y \notin \{y_i\}_{i=1}^\ell$, then remove $\{(\hat{k}, y_i)\}_{i=1}^\ell$ from Y . If $y = y_j$ for some j , then remove $\{(\hat{k}, y_i)\}_{i=1}^{j-1}$ from Y . Then take the lexicographically smallest of the remaining elements of I , put it into Y , etc. At any step of the construction, one element is added to Y and at most q are removed from I . Since I initially contains at least $\varepsilon N/2$ elements, in the end we have $|Y| \geq \lceil \varepsilon N/2q \rceil > \varepsilon N/(2q + 1)$.

Following the proof of Theorem 2.1, the permutation $F(\hat{k}, \cdot)$ is completely specified given A , descriptions of Y and the set X such that $F(\hat{k}, X) = Y$, the value of $F(\hat{k}, \cdot)$ on $\{0, 1\}^t \setminus X$, the values of G at all points, and the values of $F(k, \cdot)$ at all points for all $k \neq \hat{k}$. (We omit the details, but remark that we crucially use the fact that in the computation of $A^\tau(\hat{k}, y)$ with $y \in Y$ we are guaranteed that $Q(\hat{k}, y)$ does not occur and, in particular, A does not make a query of the form $F^{-1}(td, y')$ with $G(td) = \hat{k}$.) This requires $2 \log \binom{N}{|Y|} + \log((N - |Y|)!)$ bits plus $\log(2^t!)$ bits to specify G and $(2^t - 1) \log(2^t!)$ bits to specify $F(k, \cdot)$ for $k \neq \hat{k}$. Using an additional t bits to specify \hat{k} , as well as a bit to specify that we are in the second case, we have that τ is specified in at most the number of bits claimed.

Proceeding as in Theorem 2.1, let A be an oracle circuit of size at most $S = 2^{t/5}$ and note that A makes at most $q = 2^{t/5}$ queries to τ . Let $N = 2^t$. From the claim, we see that the fraction of $\tau \in T_t$ such that $\Pr_{k,y}[A^\tau \text{ inverts } (k, y)] \geq 2^{-t/5}$ is at most

$$\frac{2^{t+1} \binom{N}{a}}{a!} \leq 2^{-N^{3/5}/4}$$

for sufficiently large N , where $a = 2^{-t/5} 2^t / (2^{t/5+1} + 1)$ and using the fact that $a \geq N^{3/5}/4$.

Applying Lemma 3.1 (and taking into account that A here has input length $2t$, output length t , and access to three oracles some of which are functions from $\{0, 1\}^{2t}$ to $\{0, 1\}^t$), there are at most $2^{\tilde{O}(N^{1/5})}$ circuits of size S . A union bound thus shows that the probability over a random choice of $\pi \in \Pi_t$ that there *exists* a circuit of size S for which (1) holds is at most

$$2^{\tilde{O}(N^{1/5})} \cdot 2^{-N^{3/5}/4} < 2^{-N^{1/2}}$$

for all sufficiently large N . □

4. Lower bounds. In this section we prove our lower bounds.

4.1. Pseudorandom generators. We first show a lower bound for PRG constructions, defined formally as follows.

DEFINITION 4.1. *A PRG construction from a one-way permutation is an oracle procedure $G^{(\cdot)} : \{0, 1\}^\ell \rightarrow \{0, 1\}^{\ell+k}$ that expects as an oracle a permutation $\pi \in \Pi_n$. We refer to k as the stretch of G .*

We say $G^{(\cdot)}$ is an (S_p, S_g, ε) -OWP-to-PRG weak black-box construction if for every permutation π that is S_p -hard, G^π is an (S_g, ε) -secure PRG.

For any (S_p, S_g, ε) -OWP-to-PRG weak black-box construction with stretch k , we prove that unless G queries π on at least $\Omega(k/\log S_p)$ points, it is possible to derive an *unconditional* construction of a pseudorandom generator. Before giving the proof, we provide some intuition.

The basic idea of the proof is as follows. Let $t = \Theta(\log S_p)$. First, note that if G uses as an oracle $\pi \in \Pi_{t,n}$ chosen uniformly at random, then G is an (S_g, ε) -secure PRG with all but negligible probability (since a random permutation from $\Pi_{t,n}$ is S_p -hard with all but negligible probability). Now, if G queries the oracle at only a few (say, q) points, we can encode the answers to these queries in the seed of the PRG itself. Furthermore, this encoding is short since only t bits are needed to answer a query to $\pi \in \Pi_{t,n}$. We thus obtain a PRG $G' : \{0, 1\}^{\ell'} \rightarrow \{0, 1\}^{\ell+k}$ which uses no oracle at all, but which is able to simulate the computation of G when using a random permutation oracle. The desired bound comes from the fact that G' still stretches its input provided that ℓ' is smaller than $\ell + k$. But $\ell' = \ell + qt$ since qt bounds the number of bits needed to encode the t -bit answers to G 's q queries.

THEOREM 4.2. *Let $G^{(\cdot)} : \{0, 1\}^\ell \rightarrow \{0, 1\}^{\ell+k}$ be an (S_p, S_g, ε) -OWP-to-PRG weak black-box construction that makes q queries to an oracle $\pi \in \Pi_n$. If $q < k/(5 \log S_p)$, then there exists an $(S_g, \varepsilon + 2^{-S_p^2} + q^2/S_p^5)$ -secure PRG $G' : \{0, 1\}^{\ell'} \rightarrow \{0, 1\}^{\ell+k}$ with $\ell' < \ell + k$.*

Proof. Since $G^{(\cdot)}$ is an (S_p, S_g, ε) -OWP-to-PRG construction, this means that if $\pi : \{0, 1\}^n \rightarrow \{0, 1\}^n$ is S_p -hard, then for any distinguisher T of size at most S_g we have

$$\left| \Pr_{x \in U_{\ell+k}} [T(x) = 1] - \Pr_{s \in U_\ell} [T(G^\pi(s)) = 1] \right| \leq \varepsilon.$$

Let $t = 5 \log S_p$. From Corollary 2.2 we know that a random permutation $\pi \in \Pi_{t,n}$ is S_p -hard with probability greater than $1 - 2^{-2^{t/2}}$. An averaging argument thus shows that for any circuit T of size at most S_g we have

$$(4) \quad \left| \Pr_{x \in U_{\ell+k}} [T(x) = 1] - \Pr_{\substack{\pi \in \Pi_{t,n} \\ s \in U_\ell}} [T(G^\pi(s)) = 1] \right| < \varepsilon + 2^{-2^{t/2}}.$$

Recall that any $\pi \in \Pi_{t,n}$ operates only on its first t input bits; i.e., any such π satisfies $\pi(a, b) = (\hat{\pi}(a), b)$, where $\hat{\pi}$ is a permutation over $\{0, 1\}^t$. Without loss of generality, we now assume that G always queries π with strings having distinct t -prefixes. Indeed, for any G asking arbitrary queries, one can construct a \widehat{G} with essentially the same running time, such that if G asks (a, b) with a equal to the t -prefix of a previous query, \widehat{G} simulates the answer without querying π using the previously obtained value of $\hat{\pi}(a)$. In general the behavior of \widehat{G} is different from that of G , but when we restrict to $\pi \in \Pi_{t,n}$ they are equivalent.

By assumption, G queries π at most $q < k/t$ times. We now construct G' , which takes as input a seed s' of length $\ell' \stackrel{\text{def}}{=} \ell + qt < \ell + k$. Let s denote the first ℓ bits of s' , and let $y_1, \dots, y_q \in \{0, 1\}^t$ denote the remaining qt bits. We then define

$$G'(s') = G'(s, y_1, \dots, y_q) \stackrel{\text{def}}{=} G^{y_1, \dots, y_q}(s),$$

where the notation $G^{y_1, \dots, y_q}(s)$ denotes the computation of $G^{(\cdot)}(s)$ when its i th oracle query $x_i = (a_i, b_i)$ (with $|a_i| = t$) is answered with (y_i, b) . (Here we use the fact that the t -prefixes of G 's queries are distinct.)

G' stretches its input by at least one bit and requires no oracle access. Furthermore,

$$\left| \Pr_{\substack{\pi \in \Pi_{t,n} \\ s \in U_\ell}} [T(G^\pi(s)) = 1] - \Pr_{s' \in U_{\ell'}} [T(G'(s')) = 1] \right| \leq 2 \cdot \Pr_{\vec{y} \in \{0,1\}^{qt}} [\text{Coll}],$$

where we let $\vec{y} = y_1, \dots, y_q$ and Coll denotes the event that these q values are not distinct. An easy birthday problem calculation shows that $\Pr_{\vec{y} \in \{0,1\}^{qt}} [\text{Coll}] \leq q^2/2^{t+1}$, which together with (4) implies the statement of the theorem. \square

4.2. Universal one-way hash functions. In this section we prove lower bounds for constructions of universal one-way hash functions based on one-way permutations. The formal definition of such constructions follows.

DEFINITION 4.3. *A construction of a UOWHF from a one-way permutation is an oracle procedure $H^{(\cdot)}(\cdot, \cdot)$ that expects as an oracle a permutation $\pi \in \Pi_n$ and is given inputs $s \in \{0, 1\}^r$ and $x \in \{0, 1\}^{\ell+k}$. The output is $H^\pi(s, x) \in \{0, 1\}^\ell$. We refer to k as the compression of H .*

We say H is an (S_p, S_h, ε) -OWP-to-UOWHF semi-black-box construction if for every π that is S_p -hard, H^π is an (S_h, ε) -UOWHF even for circuits given oracle access to π .

We show that if there exists an (S_p, S_h, ε) -OWP-to-UOWHF semi-black-box construction with compression k making $q < k/(5 \log S_p)$ queries to its oracle π , then it is possible to derive an unconditionally secure construction of a UOWHF (i.e., without any access to π).

As in the case of PRGs, we first observe that H^π is secure when π is chosen uniformly at random from $\Pi_{t,n}$, for $t = 5 \log S$. We then show that the computation of H^π for random $\pi \in \Pi_{t,n}$ can be simulated by an H' (without any oracle access) by including as part of the key for H' the t -prefixes of the answers for the q queries H makes to π . Furthermore, we include in the output of H' the t -prefixes of the q queries themselves. The crux of the proof is to show that H' is a UOWHF. As some intuition toward this, note that since the t -prefixes of the queries (resp., answers) are included with the output (resp., key), an adversary finding a collision in H' is bound to these particular values. Hence, any collision in H' is also a collision in H^π . Since $\ell + qt < \ell + k$ (and hence H' is length-decreasing) whenever $q < k/(5 \log S_p)$, this yields the desired bound.

THEOREM 4.4. *Let $H^{(\cdot)} : \{0, 1\}^r \times \{0, 1\}^{\ell+k} \rightarrow \{0, 1\}^\ell$ be an (S_p, S_h, ε) -OWP-to-UOWHF semi-black-box construction that makes q queries to an oracle $\pi \in \Pi_n$. If $q < k/(5 \log S_p)$, then there exists an $(S_h - S_H, \varepsilon + 2^{-S_p^2} + q^2/2S_p^5)$ -secure UOWHF $H' : \{0, 1\}^{r'} \times \{0, 1\}^{\ell+k} \rightarrow \{0, 1\}^\ell$ with $\ell' < \ell + k$, where S_H is the size of the circuit computing H .*

Proof. Since $H^{(\cdot)}$ is an (S_p, S_h, ε) -OWP-to-UOWHF construction, this means that if $\pi \in \Pi_n$ is S_p -hard, then any circuit A of size $\leq S_h$ finds a collision with probability at most ε ; that is,

$$\Pr_{\substack{s \in U_r \\ z \in U_{\ell+k}}} \left[A^\pi(s, z, H^\pi(s, z)) = z' : z' \neq z \wedge H^\pi(s, z') = H^\pi(s, z) \right] \leq \varepsilon.$$

We say that z' as above is a *collision* exactly when $z' \neq z$ but $H^\pi(s, z') = H^\pi(s, z)$.

We now restrict the class of adversaries A under consideration. We consider adversaries that do not access π arbitrarily but are instead simply given the t -prefixes of the queries and answers made during the computation of $H^\pi(s, z)$. Since such restricted adversaries can be simulated by general adversaries with overhead S_H (recall that this is the size of the circuit computing H), we have that if π is S_p -hard and A is a circuit of size $\leq S_h - S_H$, then

$$\Pr_{\substack{s \in U_r \\ z \in U_{\ell+k}}} [A(s, y_1, \dots, y_q, z, H^\pi(s, z), x_1, \dots, x_q) = z' : z' \text{ is a collision}] \leq \varepsilon,$$

where x_1, \dots, x_q are the t -prefixes of the q queries made to π during computation of $H^\pi(s, z)$, and y_1, \dots, y_q are the t -prefixes of the corresponding answers.

Let $t = 5 \log S_p$. From Corollary 2.2 we know that a random permutation $\pi \in \Pi_{t,n}$ is S_p -hard with probability greater than $1 - 2^{-2^{t/2}}$. An averaging argument thus shows that for any circuit A of size $\leq S_h - S_H$ we have

$$(5) \quad \Pr_{\substack{\pi \in \Pi_{t,n} \\ s \in U_r, z \in U_{\ell+k}}} [A(s, y_1, \dots, y_q, z, H^\pi(s, z), x_1, \dots, x_q) = z' : z' \text{ is a collision}] < \varepsilon + 2^{-2^{t/2}}.$$

As in the proof of Theorem 4.2, we may assume without loss of generality that H queries π on points with distinct t -prefixes. Consider the function $H' : \{0, 1\}^{r'} \times \{0, 1\}^{\ell+k} \rightarrow \{0, 1\}^{\ell'}$ defined as follows, where $r' = r + qt$ and $\ell' = \ell + qt$:

$$H'(s', z) = H'((s, y_1, \dots, y_q), z) \stackrel{\text{def}}{=} H^{y_1, \dots, y_q}(s, z), x_1, \dots, x_q,$$

where by $H^{y_1, \dots, y_q}(s, z)$ we mean the computation of $H^{(\cdot)}(s, z)$ when its i th oracle query (a_i, b_i) (with $|a_i| = t$) is answered with (y_i, b_i) , and where x_1, \dots, x_q denote the t -prefixes of the q oracle queries made (i.e., $x_i = a_i$). Note that if $q < k/t$, then $\ell' < \ell + k$, so $H'(s', \cdot)$ is a length-decreasing function.

By definition of H' , if $H'((s, y_1, \dots, y_q), z) = H'((s, y_1, \dots, y_q), z')$, then $H^\pi(s, z) = H^\pi(s, z')$ for any π satisfying $\pi(x_1) = y_1, \dots, \pi(x_q) = y_q$. Thus,

$$\begin{aligned} & \Pr_{\substack{s' \in U_{r'} \\ z \in U_{\ell+k}}} \left[A(s', z, H'(s', z)) = z' : z' \neq z \wedge H'(s', z') = H'(s', z) \right] \\ & \leq \Pr_{\substack{\pi \in \Pi_{t,n} \\ s \in U_r, z \in U_{\ell+k}}} [A(s, y_1, \dots, y_q, z, H^\pi(s, z), x_1, \dots, x_q) = z' : z' \text{ is a collision}] \\ & \quad + \Pr_{\bar{y} \in \{0,1\}^{qt}} [\text{Coll}], \end{aligned}$$

where Coll denotes the event that the q values y_1, \dots, y_q are not distinct.⁷ Equation (5) and a simple birthday problem calculation give the result stated in the theorem. \square

⁷In fact—in contrast to the seemingly similar situation arising in the proof of Theorem 4.2—there is no need to conserve random bits here since the values y_1, \dots, y_q are included as part of the *key* and not the *output* (and the length of the key is irrelevant for our purposes). A tighter security reduction is possible by lowering $\Pr[\text{Coll}]$, for example, by including random values y_1, \dots, y_{2q} in the key and using the first q distinct values (when they exist) to answer the queries of H .

4.3. Public-key encryption. We begin with a definition of constructions of public-key encryption schemes.

DEFINITION 4.5. *A construction of a public-key encryption scheme based on trapdoor permutations is a tuple of oracle procedures $\mathcal{PK}\mathcal{E}^{(\cdot)} = (\text{Gen}^{(\cdot)}, \text{Enc}^{(\cdot)}, \text{Dec}^{(\cdot)})$ such that, for all $\tau \in T_n$, the resulting $\mathcal{PK}\mathcal{E}^\tau$ satisfies the functional definition of a public-key encryption scheme. (The construction may be for either an interactive or a noninteractive encryption scheme, as it does not affect our results.)*

We say $\mathcal{PK}\mathcal{E}^{(\cdot)}$ is an (S_p, S_e, ε) -TDP-to-PKE weak black-box construction if for every oracle $\tau \in T_n$ that is S_p -trapdoor one way, $\mathcal{PK}\mathcal{E}^\tau$ is (S_e, ε) -secure.

We prove that for any such construction which encrypts messages of length m , unless Enc^τ queries τ at least $\Omega(m/\log S_p)$ times, there exists a one-way function which does not require any oracle access. Our proof proceeds by showing that unless Enc^τ makes at least $\Omega(m/\log S_p)$ queries to τ , we can explicitly construct an interactive, private-key encryption scheme $(\text{Enc}', \text{Dec}')$ requiring no access to the oracle and in which the encrypted message is longer than the shared key. Using a previous result of Impagliazzo and Luby [30] (see also Lemma 4.6), this implies the existence of a one-way function.

As in the previous proofs, we first observe that a random $\tau \in T_{t,n}$ is S_p -hard with all but negligible probability when $t = 5 \log S_p$ (cf. Corollary 2.4). To construct an interactive, private-key encryption scheme without access to an oracle, we have the parties simulate a random τ by appropriately choosing random t -prefixes for the answers to their queries, as needed. The bits to simulate τ cannot be included in the private key, since the encryption and decryption algorithms may make their queries in different order and, indeed, may make different queries altogether. However, we must somehow ensure consistency between the oracle answers of the sender and the receiver. A possibility that comes to mind is to have each party include, along with each protocol message it sends to the other party, a list of t -prefixes for the queries and answers generated thus far in accessing τ . In this case, however, privacy is no longer guaranteed as the queries may reveal information about the plaintext message. But this is easily remedied in the private-key setting: the parties simply share a sufficiently long one-time pad in advance and then encrypt their queries and answers using this pad.

Let q_g be the number of queries made to τ by Gen , and let q_e be the number of queries made by Enc . The private-key encryption scheme outlined above requires a shared key of size roughly $O(t) \cdot (q_g + q_e)$ to encrypt an m -bit message. Recalling the result of Impagliazzo and Luby [30], if $O(t) \cdot (q_g + q_e) < m$, then the key is shorter than the message and a one-way function exists. This already gives a weak lower bound. To obtain the better lower bound $q_e < m/O(t)$ (so that we bound the efficiency of encryption alone), additional work is needed; details are given in the proof of Theorem 4.7.

We begin by showing that the existence of a private-key encryption scheme (Enc, Dec) which securely encrypts messages longer than the shared key implies the existence of a one-way function. Although this result is already known [30] (without the concrete bounds given below), we give a much simpler and more direct proof. We stress that the result applies even in the case of interactive encryption.

LEMMA 4.6. *Let (Enc, Dec) be an (S, ε) -secure private-key encryption scheme for messages of length m using a shared key of length $k < m$. Let S_{Enc} be the size of the circuit needed to run the encryption protocol (i.e., the size of the circuit for Enc in the noninteractive case, or the combined sizes of the circuits for Enc and Dec*

in the interactive setting). Then for any $\ell \in \mathbb{N}$ there exists a function f which is $(S - \ell S_{\text{Enc}}, \ell\varepsilon + 2^{-\ell(m-k)})$ -one way.

Proof. We prove the lemma for the case of interactive encryption, which implies the same result for the degenerate, noninteractive case as well. First note that via standard hybrid argument, running ℓ parallel copies of (Enc, Dec) using ℓ independent keys yields an $(S, \ell\varepsilon)$ -secure private-key encryption scheme for messages of length ℓm in which the shared key has length ℓk . Let $\mathcal{SK}\mathcal{E}_\ell = (\text{Enc}_\ell, \text{Dec}_\ell)$ denote this modified scheme.

Define f by $f(sk, M, \omega_1, \omega_2) = \widehat{\text{Enc}}_\ell(sk, M; \omega_1, \omega_2) \| M$, where ω_1, ω_2 represent the random coins used by Enc_ℓ and Dec_ℓ , respectively. We claim that this function is (S', ε') -one way, where $S' = S - \ell S_{\text{Enc}}$ and $\varepsilon' = \ell\varepsilon + 2^{-\ell(m-k)}$. If not, then there is an algorithm B of size at most S' for which $\text{Succ}_{B,f} > \varepsilon'$, where

$$\text{Succ}_{B,f} \stackrel{\text{def}}{=} \Pr \left[sk \leftarrow \{0, 1\}^{\ell k}; M \leftarrow \{0, 1\}^{\ell m}; T \leftarrow \widehat{\text{Enc}}_\ell(sk, M) : B(T \| M) \in f^{-1}(T \| M) \right].$$

We show how such a B can be used to construct an algorithm A of size at most S for which $\text{Succ}_{A, \mathcal{SK}\mathcal{E}_\ell} > \ell\varepsilon$, where

$$\text{Succ}_{A, \mathcal{SK}\mathcal{E}_\ell} \stackrel{\text{def}}{=} \left| \Pr_{\substack{M_0, M_1 \in \{0, 1\}^{\ell m} \\ T \in \mathcal{SK}\mathcal{E}_\ell(M_0)}} [A(M_0, M_1, T) = 1] - \Pr_{\substack{M_0, M_1 \in \{0, 1\}^{\ell m} \\ T \in \mathcal{SK}\mathcal{E}_\ell(M_1)}} [A(M_0, M_1, T) = 1] \right|.$$

This implies that there exist two messages M_0, M_1 for which A can distinguish encryptions of M_0 from encryptions of M_1 with probability better than $\ell\varepsilon$, contradicting the assumed security of $(\text{Enc}_\ell, \text{Dec}_\ell)$. Thus, we are done once we have demonstrated such an A .

Define A as follows. On input (M_0, M_1, T) , algorithm A runs $B(T \| M_0)$ to obtain the result $sk' \| M' \| \omega'_1 \| \omega'_2$. It then checks whether $f(sk', M', \omega'_1, \omega'_2) \stackrel{?}{=} T \| M_0$. If so (i.e., B has succeeded in inverting f), then A outputs 0. Otherwise, A outputs 1. Note that $|A| = |B| + \ell S_{\text{Enc}} \leq S$, as required.

First, note that

$$\Pr_{\substack{M_0, M_1 \in \{0, 1\}^{\ell m} \\ T \in \mathcal{SK}\mathcal{E}_\ell(M_0)}} [A(M_0, M_1, T) = 0] = \text{Succ}_{B,f} > \varepsilon'.$$

For a transcript T , we say (sk, M) is *consistent with* T if there exist ω_1, ω_2 such that $T = \widehat{\text{Enc}}_\ell(sk, M; \omega_1, \omega_2)$. We have

$$\begin{aligned} & \Pr_{\substack{M_0, M_1 \in \{0, 1\}^{\ell m} \\ T \in \mathcal{SK}\mathcal{E}_\ell(M_1)}} [A(M_0, M_1, T) = 1] \\ & \leq \Pr[sk \leftarrow \{0, 1\}^{\ell k}; M_0, M_1 \leftarrow \{0, 1\}^{\ell m}; \\ & \quad T \leftarrow \widehat{\text{Enc}}_\ell(sk, M_1) : \exists sk' \text{ s.t. } (sk', M_0) \text{ is consistent with } T] \\ & \leq \sum_{sk' \in \{0, 1\}^{\ell k}} \Pr[sk \leftarrow \{0, 1\}^{\ell k}; M_0, M_1 \leftarrow \{0, 1\}^{\ell m}; \\ & \quad T \leftarrow \widehat{\text{Enc}}_\ell(sk, M_1) : (sk', M_0) \text{ is consistent with } T]. \end{aligned}$$

Perfect correctness of the encryption scheme implies that for any sk, T there is at most one value of $M \in \{0, 1\}^{\ell m}$ such that (sk, M) is consistent with T . Using this and the fact that M_0 is chosen at random independent of anything else gives

$$\begin{aligned} \Pr_{\substack{M_0, M_1 \in \{0, 1\}^{\ell m} \\ T \in \mathcal{SK}\mathcal{E}_\ell(M_1)}} [A(M_0, M_1, T) = 1] &\leq \sum_{sk' \in \{0, 1\}^{\ell k}} 2^{-\ell m} \\ &= 2^{\ell(k-m)}. \end{aligned}$$

Putting everything together shows that $\text{Succ}_{A, \mathcal{SK}\mathcal{E}_\ell} > \varepsilon' - 2^{\ell(k-m)} \geq \ell\varepsilon$, giving the desired contradiction. \square

We remark that an analogue of the above lemma is known to hold also for the case of (interactive) private-key encryption schemes *with error* [30]. This, in turn, implies results analogous to those of Theorems 4.7 and 4.9 for black-box constructions of encryption schemes with error. However, as we were unable to simplify the proof of [30] in this setting (and as the concrete bounds on the resulting one-way function are rather unwieldy), we do not explicitly focus on the case of encryption schemes with error here.

Our main result of this section follows. The theorem is stated for the case of noninteractive public-key encryption, but the proof immediately extends to the case of *interactive* public-key encryption as well (where q_e will in this case refer to the total number of queries made by sender and receiver during the encryption protocol, and S_{Enc} will refer to the sizes of Enc and Dec jointly).

THEOREM 4.7. *Let $\mathcal{PK}\mathcal{E}^{(\cdot)} = (\text{Gen}^{(\cdot)}, \text{Enc}^{(\cdot)}, \text{Dec}^{(\cdot)})$ be an (S_p, S_e, ε) -TDP-to-PKE weak black-box construction for messages of length m , and let $t = 5 \log S_p$. Assume Gen makes q_g queries to an oracle $\tau \in T_n$ and Enc makes q_e queries to τ ; set $\ell = 2 \cdot \lceil 5tq_g / (m - 5tq_e) \rceil$. Assume further that $\varepsilon < (1/4 - 2^{-S_p^2})/\ell$. If $q_e < m/5t$, then there exists an $(S_e - 3\ell S_{\text{Enc}}, 3/4)$ -one-way function (without access to any oracle), where S_{Enc} is the size of the circuit for Enc.*

Proof. Note that we did not try to optimize the constants in the proof or the required bound on ε . In applications of cryptographic interest, S_p and S_e are typically superpolynomial, S_{Enc} , q_g , q_e , and m are (small) polynomials, and ε is negligible; thus, $\varepsilon \ll (1/4 - 2^{-S_p^2})/\ell$ and $S_e \gg 3\ell S_{\text{Enc}}$ anyway.

Let $\mathcal{PK}\mathcal{E}^{(\cdot)} = (\text{Gen}^{(\cdot)}, \text{Enc}^{(\cdot)}, \text{Dec}^{(\cdot)})$. As in the proof of Lemma 4.6, for any $\ell \in \mathbb{N}$ we may construct a public-key encryption scheme $\mathcal{PK}\mathcal{E}_\ell^{(\cdot)} = (\text{Gen}_\ell^{(\cdot)}, \text{Enc}_\ell^{(\cdot)}, \text{Dec}_\ell^{(\cdot)})$ for ℓm -bit messages in the natural way; furthermore, we may set $\text{Gen}_\ell = \text{Gen}$ since we are now in the public-key setting and so key generation need be done only once. It is easy to see (via hybrid argument) that $\mathcal{PK}\mathcal{E}_\ell^{(\cdot)}$ is an $(S_p, S_e - \ell S_{\text{Enc}}, \ell\varepsilon)$ -TDP-to-PKE construction, where the number of queries made by Gen_ℓ is q_g and the number of queries made by Enc_ℓ is at most ℓq_e .

Set $\ell = 2 \cdot \lceil 5tq_g / (m - 5tq_e) \rceil$ as in the statement of the theorem, and let $S' = S_e - \ell S_{\text{Enc}}$ and $\varepsilon' = \ell\varepsilon + 2^{-S_p^2}$. We use $\mathcal{PK}\mathcal{E}_\ell^{(\cdot)}$ to construct an (S', ε') -secure interactive, private-key encryption scheme $\mathcal{SK}\mathcal{E} = (\text{Enc}', \text{Dec}')$ for ℓm -bit messages in which the shared key will have length $5t \cdot (q_g + \ell q_e)$. Furthermore, $\mathcal{SK}\mathcal{E}$ will require no access to the trapdoor permutation oracle. Finally, we have $5t \cdot (q_g + \ell q_e) < \ell m$ (in fact, $\ell m - 5t \cdot (q_g + \ell q_e) \geq 1$) and $\varepsilon' < 1/4$; thus, application of Lemma 4.6 (with $\ell = 1$ there) yields the desired result.

Security of $\mathcal{PK}\mathcal{E}_\ell^{(\cdot)}$ implies that if τ is S_p -hard, then for any circuit B of size $\leq S'$

and for any messages $M_0, M_1 \in \{0, 1\}^{\ell m}$ we have

$$\left| \Pr_{v \in \mathcal{PK}\mathcal{E}_\ell^\tau(M_0)} [B(v) = 1] - \Pr_{v \in \mathcal{PK}\mathcal{E}_\ell^\tau(M_1)} [B(v) = 1] \right| \leq \ell\varepsilon.$$

Corollary 2.4 shows that a random $\tau \in T_{t,n}$ is S_p -hard except with probability less than $2^{-S_p^2}$. A straightforward averaging argument thus shows that for any circuit B of size $\leq S'$ and for any messages $M_0, M_1 \in \{0, 1\}^{\ell m}$ we have

$$(6) \quad \left| \Pr_{\substack{\tau \in T_{t,n} \\ v \in \mathcal{PK}\mathcal{E}_\ell^\tau(M_0)}} [B(v) = 1] - \Pr_{\substack{\tau \in T_{t,n} \\ v \in \mathcal{PK}\mathcal{E}_\ell^\tau(M_1)}} [B(v) = 1] \right| < \ell\varepsilon + 2^{-S_p^2} = \varepsilon'.$$

As mentioned in the discussion at the beginning of this section, our private-key encryption scheme $\mathcal{SK}\mathcal{E}$ will simulate a random $\tau \in T_{t,n}$ for algorithms Gen , Enc_ℓ , and Dec_ℓ . We achieve this simulation using a simulation procedure \mathcal{SIM} which ensures consistency of the answers to all oracle queries. This procedure takes as input a list L of (appropriate prefixes of) previous oracle queries and answers; before answering any query, \mathcal{SIM} examines L and ensures that the answer it gives will not generate any inconsistencies. After answering a query, L is updated accordingly. As an example, if query $td\|b$ to G was answered by $k\|b$ (where $|td| = |k| = t$), then subsequent query $td'\|b'$ to G must be answered by $k'\|b'$ where $k' = k$ iff $td' = td$. A more involved procedure is needed to answer queries to F and F^{-1} . We now describe the details of this simulation.

$\mathcal{SIM}(L)$.

- On query $G(td\|b)$ (where $|td| = t$): if $\exists k$ s.t. $(td, k) \in L$, return $k\|b$. Otherwise pick random $k \in \{0, 1\}^t$ such that $\forall td' : (td', k) \notin L$, return $k\|b$, and store (td, k) in L .
- On query $F(k\|b, x\|b')$ (where $|k| = |x| = t$):
 1. if $\exists y$ s.t. $(k, x, y) \in L$, return $y\|b'$.
 2. Otherwise, if $\exists td$ s.t. $(td, k) \in L$, choose random $y \in \{0, 1\}^t$ such that $\forall x' : (k, x', y) \notin L$, return $y\|b'$, and store (k, x, y) in L .
 3. Otherwise, choose random $td \in \{0, 1\}^t$ such that $\forall k' : (td, k') \notin L$, choose random $y \in \{0, 1\}^t$, return $y\|b'$, and store $(k, td), (k, x, y)$ in L .
- On query $F^{-1}(td\|b, y\|b')$ (where $|tk| = |y| = t$):
 1. if $\exists k, x$ s.t. $(td, k), (k, x, y) \in L$, return $x\|b'$
 2. Otherwise, if $\exists k$ s.t. $(td, k) \in L$, choose random $x \in \{0, 1\}^t$ such that $\forall y' : (k, x, y') \notin L$, return $x\|b'$, and store (k, x, y) in L
 3. Otherwise, choose random $k \in \{0, 1\}^t$ such that $\forall td' : (td', k) \notin L$, choose random $x \in \{0, 1\}^t$, return $x\|b'$, and store $(td, k), (k, x, y)$ in L

Note that each time a query is answered, at most $5t$ bits are stored in L .

Construct $\mathcal{SK}\mathcal{E}$ as follows. Parse the shared key s as (s_1, s_2) , where $|s_1| = 5tq_g$ and $|s_2| = 5tq_e$. To encrypt message M , the receiver Dec' begins by initializing list $L := \emptyset$. The receiver then computes $(pk, sk) \leftarrow \text{Gen}^{\mathcal{SIM}(L)}$ (updating L in the process) and sends $pk, s_1 \oplus L$ to the sender. The receiver stores sk, L for later use. On receiving the first message pk, \hat{s}_1 , the sender computes $L_1 := s_1 \oplus \hat{s}_1$ and sets $L := L_1$. The sender then computes $C \leftarrow \text{Enc}_\ell^{\mathcal{SIM}(L)}(pk, M)$ and sets $L_2 := L \setminus L_1$. Finally, Enc' sends $C, s_2 \oplus L_2$ to the receiver. On receiving message C, \hat{s}_2 , the receiver decrypts by setting $L_2 := s_2 \oplus \hat{s}_2$ and $L := L_0 \cup L_2$ (here, L_0 is the list stored by the receiver from the first stage). The receiver can then compute $M := \text{Dec}_\ell^{\mathcal{SIM}(L)}(sk, C)$.

It is clear that $\mathcal{SK}\mathcal{E}$ has correct decryption. We now show that the scheme is (S', ε') -secure. Assume toward a contradiction that there exists a circuit A of size $\leq S'$ and messages $M_0, M_1 \in \{0, 1\}^{\ell m}$ such that

$$\left| \Pr_{v \in \mathcal{SK}\mathcal{E}(M_0)} [A(v) = 1] - \Pr_{v \in \mathcal{SK}\mathcal{E}(M_1)} [A(v) = 1] \right| > \varepsilon'.$$

We construct circuit B attacking $\mathcal{PK}\mathcal{E}_\ell$ as follows. On input pk, C , circuit B picks random strings \hat{s}_1, \hat{s}_2 , where $|\hat{s}_1| = 5tq_g$ and $|\hat{s}_2| = 5t\ell q_e$. Then, B outputs $A(pk, \hat{s}_1, C, \hat{s}_2)$. Because the keys s_1, s_2 of $\mathcal{SK}\mathcal{E}$ are used as a one-time pad, it is easy to see that, for $b \in \{0, 1\}$,

$$\Pr_{\substack{\tau \in T_{t,n} \\ (pk, C) \in \mathcal{PK}\mathcal{E}_\ell^\tau(M_b)}} [B(pk, C) = 1] = \Pr_{v \in \mathcal{SK}\mathcal{E}(M_b)} [A(v) = 1].$$

Thus, the advantage of B is equal to the advantage of A (which is greater than ε'), contradicting (6). \square

4.4. Private-key encryption. The techniques of the previous section can be adapted to show a similar lower bound for private-key encryption schemes based on trapdoor permutations; note that trapdoor permutations generalize one-way permutations (which are sufficient for private-key encryption), and therefore our result shows that improved efficiency cannot be obtained in this case even by assuming a stronger primitive.

DEFINITION 4.8. *A construction of a private-key encryption scheme based on trapdoor permutations is a pair of oracle procedures $\mathcal{SK}\mathcal{E}^{(\cdot)} = (\text{Enc}^{(\cdot)}, \text{Dec}^{(\cdot)})$ such that for all $\tau \in T_n$, the resulting $\mathcal{SK}\mathcal{E}^\tau$ satisfies the functional definition of a private-key encryption scheme given earlier. (Again, the construction may yield either an interactive or a noninteractive scheme.)*

We say $\mathcal{SK}\mathcal{E}^{(\cdot)}$ is an (S_p, S_e, ε) -TDP-to-SKE weak black-box construction if for every $\tau \in T_n$ that is S_p -hard, $\mathcal{SK}\mathcal{E}^\tau$ is (S_e, ε) -secure.

Suppose a construction of the above type exists which encrypts m -bit messages using a shared key of length k . We show that unless Enc^τ queries τ at least $q = \Omega(\frac{m-k}{\log S_p})$ times, then an unconditional one-way function exists. This matches the known upper bound, even for schemes constructed using one-way permutations.

The proof is similar to that of Theorem 4.7 in that we convert $\mathcal{SK}\mathcal{E}^{(\cdot)}$ to a private-key encryption scheme $\mathcal{SK}\mathcal{E}'$ that does not access an oracle at all. The only difference between the proof here and the proof of Theorem 4.7 is that here the parties need to share a k -bit key in addition to the one-time pad used to encrypt their simulated queries and answers to the oracle. Set $t = 5 \log S_p$. The resulting $\mathcal{SK}\mathcal{E}'$ requires a shared key of length $k + 5tq$ and encrypts an m -bit message. As before, then, if $k + 5tq < m$ we obtain a private-key encryption scheme (making no oracle queries) in which the message is longer than the key. By Lemma 4.6, this implies the existence of a one-way function.

The following theorem is stated for the case of noninteractive private-key encryption, but the proof immediately extends to the case of interactive private-key encryption as well.

THEOREM 4.9. *Let $\mathcal{SK}\mathcal{E}^{(\cdot)} = (\text{Enc}^{(\cdot)}, \text{Dec}^{(\cdot)})$ be an (S_p, S_e, ε) -TDP-to-SKE weak black-box construction for messages of length m using a key of length k in which Enc makes q queries to an oracle $\tau \in T_n$. Let $t = 5 \log S_p$. If $q < \frac{m-k}{5t}$, then there exists*

an $(S_e, \varepsilon + 2^{-S_p^2})$ -secure private-key encryption scheme in which the message is longer than the key, without access to any oracle.

Proof. The proof is substantially similar to the proof of Theorem 4.7, so the discussion here will be somewhat terse. Let $t = 5 \log S_p$. As in the previous proof, security of the given construction and a straightforward averaging argument imply that for any circuit B of size $\leq S_e$ and for any messages $M_0, M_1 \in \{0, 1\}^m$ we have

$$\left| \Pr_{\substack{\tau \in T_{t,n} \\ v \in \text{SK}\mathcal{E}^\tau(M_0)}} [B(v) = 1] - \Pr_{\substack{\tau \in T_{t,n} \\ v \in \text{SK}\mathcal{E}^\tau(M_1)}} [B(v) = 1] \right| < \varepsilon + 2^{-S_p^2}.$$

We now construct an $(S_e, \varepsilon + 2^{-S_p^2})$ -secure private-key encryption scheme $\text{SK}\mathcal{E}' = (\text{Enc}', \text{Dec}')$ for m -bit messages in which the shared key has length $k' \stackrel{\text{def}}{=} k + 5t \cdot q$. Furthermore, $\text{SK}\mathcal{E}'$ requires no oracle access. If $q < (m - k)/5t$, then $k' < m$ and we obtain the desired result.

The approach for constructing $\text{SK}\mathcal{E}'$ is exactly as in Theorem 4.7 (and as discussed earlier in this section), and in particular we use the same simulation procedure SIM as there. $\text{SK}\mathcal{E}'$ is constructed as follows. The shared key s of length k' is parsed as (s_1, s_2) , where $|s_1| = k$ and $|s_2| = 5t \cdot q$. To encrypt message M , the sender initializes $L := \emptyset$, computes $C \leftarrow \text{Enc}^{\text{SIM}(L)}(s_1, M)$ (updating L in the process), and then sends $C, s_2 \oplus L$ to the receiver. The receiver obtains ciphertext C, \hat{L} , recovers $L = \hat{L} \oplus s_2$, and then computes $M = \text{Dec}^{\text{SIM}(L)}(s_1, C)$.

It is not hard to see that $\text{SK}\mathcal{E}'$ has correct decryption. Arguing exactly as in Theorem 4.7 we see that $\text{SK}\mathcal{E}'$ is $(S_e, \varepsilon + 2^{-S_p^2})$ -secure, completing the proof of the theorem. \square

4.5. Signature schemes. We now demonstrate a lower bound on the efficiency of signature verification for any signature scheme based on one-way permutations.

DEFINITION 4.10. *A construction of a digital signature scheme based on one-way permutations is a tuple of procedures $\text{SIG}^{(\cdot)} = (\text{Gen}^{(\cdot)}, \text{Sign}^{(\cdot)}, \text{Vrfy}^{(\cdot)})$ such that for all $\pi \in \Pi_n$, the resulting SIG^π satisfies the functional definition of a signature scheme given earlier. We say $\text{SIG}^{(\cdot)}$ is an $(S_p, S_\Sigma, \varepsilon)$ -OWP-to-signature semi-black-box construction if for every oracle $\pi \in \Pi_n$ that is S_p -hard, SIG^π is (S_Σ, ε) -secure, where this must hold even for circuits given access to π .*

Given a construction of this sort, we prove that unless Vrfy queries π at least $\Omega(m/\log S_p)$ times, then it is possible to construct from SIG a one-way function which does not access any oracle. Note that this one-way function could then be used to construct a secure signature scheme (which requires no oracle access) [38].

We start with an informal overview of our proof technique. As a first attempt to construct a one-way function from the verification algorithm, one might define

$$F_1(PK, M, \sigma) = PK \parallel \text{Vrfy}^{(\cdot)}(PK, M, \sigma).$$

Intuitively, this function is difficult to invert on elements of the form $PK \parallel 1$ if PK is a valid and randomly generated public key, since inverting the function on points of this form is equivalent to signature forgery. As presently defined, however, evaluating F_1 requires calls to π ; however, our goal is to construct a function which does not require access to any oracle. As in the previous section, however, one may observe that π is S_p -hard (and thus SIG is secure) when π is uniformly chosen from $\Pi_{t,n}$ for $t = 5 \log S_p$ (cf. Corollary 2.2). So, if Vrfy makes q queries to π , then specifying qt

bits as the answers to these queries removes any need to query the oracle. Based on this, one might consider the function

$$(7) F_2(PK, y_1, \dots, y_q, M, \sigma) = PK \|y_1\| \cdots \|y_q\| x_1 \| \cdots \|x_q\| \text{Vrfy}^{y_1, \dots, y_q}(PK, M, \sigma),$$

where $|y_1| = \cdots = |y_q| = |x_1| = \cdots = |x_q| = t$ and the i th query $x_i \|b_i$ of Vrfy is answered with $y_i \|b_i$. (We assume without loss of generality that Vrfy queries π with strings having distinct t -prefixes.) The intuition, as before, is that F_2 is difficult to invert on elements of the form $PK \|y\| \vec{x} \|1$ if PK, \vec{y} , and \vec{x} are chosen appropriately.

Now, however, another problem arises. For F_2 to qualify as a one-way function, it must be hard to invert $F_2(PK, \vec{y}, M, \sigma)$ when PK, \vec{y}, M, σ are sampled from an *efficiently sampleable* distribution (cf. Lemma 4.12). More specifically, a proof of one-wayness will need to show how to efficiently sample PK, \vec{y}, M, σ such that inverting the value $F_2(PK, \vec{y}, M, \sigma)$ results in a signature forgery and hence a contradiction. A necessary condition for inversion to result in signature forgery is that $\text{Vrfy}^{\vec{y}}(PK, M, \sigma) = 1$. Generating PK, \vec{y}, M, σ such that this holds is easy if we have the secret key SK , but then inverting F_2 and forging a signature does not yield the desired contradiction!

Instead, in the proof we will obtain M, σ from the signer. In this case, however, inverting $F_2(PK, \vec{y}, M, \sigma)$ does not result in a forgery if it results in the same message/signature pair M, σ . We come now to the crux of our proof. If the number of queries is small, we show that inversion of F_2 yields a different message M' (and thus a successful forgery) with noticeable probability. More precisely, for randomly generated $PK, \vec{y}, \vec{x}, \sigma$, let

$$Y = PK \|y\| \vec{x} \|1 = F_2(PK, \vec{y}, M, \sigma).$$

If $|\vec{x}| = \sum_{i=1}^q |x_i| < |M|$, then (on average) there exists an element $PK \|y\| M' \| \sigma' \in F_2^{-1}(Y)$ with $M' \neq M$; hence inverting Y results in a forgery with noticeable probability. This idea is formalized in the proof of the following theorem.

THEOREM 4.11. *Let $STG^{(\cdot)}$ be an $(S_p, S_\Sigma, \varepsilon)$ -OWP-to-signature semi-black-box construction for messages of length m in which Vrfy makes q queries to an oracle $\pi \in \Pi_n$, algorithms Gen , Sign , and Vrfy jointly make \hat{q} queries to π , and $\varepsilon < 1/4 - 2^{-S_p^2}$. If $q < m/(5 \log S_p)$, then there exists an $(S_\Sigma - S_{\text{Gen}} - S_{\text{Sign}} - 2S_{\text{Vrfy}}, 3/4 + \hat{q}^2/2S_p^5)$ -one-way function (without access to a permutation oracle), where $S_{\text{Gen}}, S_{\text{Sign}},$ and S_{Vrfy} are the sizes of the circuits for $\text{Gen}, \text{Sign},$ and Vrfy , respectively.*

Proof. As in Theorem 4.7, we did not try to optimize the constants in the proof or the required bound on ε . In applications of cryptographic interest, one will anyway have $S_\Sigma \gg S_{\text{Gen}} + S_{\text{Sign}} + 2S_{\text{Vrfy}}$ and $\varepsilon \ll 1/4 - 2^{-S_p^2}$.

We first present a technical lemma showing that the existence of a function which is one-way over an efficiently sampleable domain implies the existence of a function which is one-way under the definition of section 2.1.

LEMMA 4.12. *Let \mathcal{D} be a distribution sampleable by a circuit of size $S_{\mathcal{D}}$ and let f be a function such that for every circuit A of size $\leq S$ we have*

$$\Pr[x \leftarrow \mathcal{D} : A(f(x)) \in f^{-1}(f(x))] \leq \delta.$$

Then there exists a function \hat{f} that is $(S - S_{\mathcal{D}}, \delta)$ -one way.

Proof of Lemma 4.12. We equate the distribution \mathcal{D} with the circuit of size $S_{\mathcal{D}}$ which samples it; i.e., $\{\mathcal{D}(r)\} \equiv \mathcal{D}$ (where r is a string of the appropriate length chosen uniformly at random). Define $\hat{f}(r) \stackrel{\text{def}}{=} f(\mathcal{D}(r))$. We claim that \hat{f} is $(S - S_{\mathcal{D}}, \delta)$ -one

way. Assume the contrary. Then there exists a circuit \hat{A} of size at most $S - S_{\mathcal{D}}$ for which

$$\Pr_r[\hat{A}(\hat{f}(r)) \in \hat{f}^{-1}(\hat{f}(r))] > \delta.$$

Toward a contradiction, define a circuit A as follows: $A(y) \stackrel{\text{def}}{=} \mathcal{D}(\hat{A}(y))$. Notice that $|A| \leq S$. Furthermore,

$$\begin{aligned} \Pr[x \leftarrow \mathcal{D} : A(f(x)) \in f^{-1}(f(x))] \\ &= \Pr_r[x := \mathcal{D}(r) : f(A(f(x))) = f(x)] \\ &= \Pr_r[f(A(\hat{f}(r))) = \hat{f}(r)] \\ &= \Pr_r[\hat{f}(\hat{A}(\hat{f}(r))) = \hat{f}(r)] \\ &= \Pr_r[\hat{A}(\hat{f}(r)) \in \hat{f}^{-1}(\hat{f}(r))] > \delta. \end{aligned}$$

The proof of the theorem proceeds by using \mathcal{SIG} to construct a function F along with a distribution \mathcal{D} such that for every circuit A of size $\leq S_{\Sigma} - S_{\text{Vrfy}}$ we have

$$(8) \quad \begin{aligned} \Pr[X \leftarrow \mathcal{D} : A(F(X)) \in F^{-1}(F(X))] &\leq \varepsilon + 2^{-S_p} + 1/2 + \hat{q}^2/2S_p^5 \\ &< 3/4 + \hat{q}^2/2S_p^5. \end{aligned}$$

Furthermore, \mathcal{D} will be computable by a circuit of size $S_{\text{Gen}} + S_{\text{Sign}} + S_{\text{Vrfy}}$. Applying Lemma 4.12 then yields the desired result.

Since $\mathcal{SIG}^{(\cdot)}$ is an $(S_p, S_{\Sigma}, \varepsilon)$ -OWP-to-signature construction, if π is S_p -hard, then for any circuit B of size $\leq S_{\Sigma}$ we have $\text{Succ}_{\pi, B} \leq \varepsilon$ where

$$\begin{aligned} \text{Succ}_{\pi, B} &\stackrel{\text{def}}{=} \\ \Pr[(PK, SK) \leftarrow \text{Gen}^{\pi}; M \leftarrow \{0, 1\}^m; \sigma \leftarrow \text{Sign}^{\pi}(SK, M); (M', \sigma') := B^{\pi}(PK, M, \sigma) : \\ &\quad \text{Vrfy}^{\pi}(PK, M', \sigma') = 1 \wedge M' \neq M]. \end{aligned}$$

Let $t = 5 \log S_p$. Corollary 2.2 shows that a random $\pi \in \Pi_{t, n}$ is S_p -hard except with probability less than $2^{-S_p^2}$. An averaging argument then implies that for any circuit B of size $\leq S_{\Sigma}$ we have $\text{Succ}_B^* < \varepsilon + 2^{-S_p^2}$ where Succ_B^* is defined analogously to $\text{Succ}_{\pi, B}$ except that the probability is now taken over random choice of $\pi \in \Pi_{t, n}$ as well.

Define a function F as in (7), repeated here for convenience:

$$F(PK, \vec{y}, M, \sigma) = PK \parallel \vec{y} \parallel \vec{x} \parallel \text{Vrfy}^{\vec{y}}(PK, M, \sigma),$$

where $\vec{y} = (y_1, \dots, y_q)$, $\vec{x} = (x_1, \dots, x_q)$, $|y_1| = \dots = |y_q| = |x_1| = \dots = |x_q| = t$, and the i th query $x_i \parallel b_i$ of Vrfy is answered with $y_i \parallel b_i$. (As in the proof of Theorem 4.2, we assume Vrfy queries its oracle on points having distinct t -prefixes.) We also define distribution \mathcal{D} by the following experiment, which depends on uniformly distributed coins r_g, r_s (of some appropriate length), $r_y \in \{0, 1\}^{\hat{q}t}$ (parsed as a sequence of t -bit strings $\hat{y}_1, \dots, \hat{y}_{\hat{q}} \in \{0, 1\}^t$), and $M \in \{0, 1\}^m$:

$$\left\{ \begin{array}{l} (PK, SK) := \text{Gen}(r_g); \\ \sigma := \text{Sign}(SK, M; r_s); \text{Vrfy}(PK, M, \sigma) \end{array} : PK \parallel \vec{y} \parallel M \parallel \sigma \right\}.$$

In the above experiment, the coins $r_y = \hat{y}_1, \dots, \hat{y}_{\hat{q}}$ are used to give a simulation⁸ of a random $\pi \in \Pi_{t,n}$ which is consistent across the executions of **Gen**, **Sign**, and **Vrfy**. The component y_i of \vec{y} is the t -prefix of the answer given in response to the i th query of **Vrfy**. Note that \mathcal{D} is computable by a circuit of size essentially $S_{\text{Gen}} + S_{\text{Sign}} + S_{\text{Vrfy}}$.

We claim that F satisfies the requirement expressed in (8). Assume toward a contradiction that there exists a circuit A of size $\leq S_{\Sigma} - S_{\text{Vrfy}}$ for which (8) does not hold. We use A to construct an algorithm B which—given PK , a random message M , and a signature σ on M —forges a valid signature on a new message M' with high probability. $B^{\pi}(PK, M, \sigma)$ first runs **Vrfy**(PK, M, σ), answering the queries of **Vrfy** by forwarding them to π . Let \vec{x} be the t -prefixes of the queries made by **Vrfy**, and let \vec{y} be the t -prefixes of the corresponding answers. Define $X = PK \parallel \vec{y} \parallel M \parallel \sigma$ and $Y = PK \parallel \vec{y} \parallel \vec{x} \parallel 1$; note that $Y = F(X)$. Finally, algorithm B computes $PK' \parallel \vec{y}' \parallel M' \parallel \sigma' = A(Y)$ and outputs (M', σ') . We clearly have $|B| \leq S_{\Sigma}$.

B outputs a successful forgery if both $F(PK' \parallel \vec{y}' \parallel M' \parallel \sigma') = Y$ and $M' \neq M$ hold. To see this, note that the first condition implies $\vec{y} = \vec{y}'$, and hence $\text{Vrfy}^{\vec{y}}(PK, M', \sigma') = 1$, and furthermore **Vrfy** makes queries with t -prefixes \vec{x} . Thus, $\text{Vrfy}^{\pi}(PK, M', \sigma') = 1$. If furthermore $M' \neq M$, then (M', σ') is a successful forgery. Finally, the distribution on X —over random choice of $\pi \in \Pi_{t,n}$ —is statistically close to distribution \mathcal{D} , where the difference is due to the fact that the $\hat{y}_1, \dots, \hat{y}_{\hat{q}}$ used in the experiment defining \mathcal{D} may not be distinct. This accounts for the term $\hat{q}^2/2S_p^2$ in the analysis below (obtained using a simple birthday problem calculation), but see footnote 7.

Let Eq be the event that $M' = M$. Then

$$\begin{aligned}
 \text{Succ}_B^* &\geq \Pr_{X \leftarrow \mathcal{D}}[Y = F(X); X' = A(Y) : X' \in F^{-1}(Y) \wedge \overline{\text{Eq}}] - \hat{q}^2/2S_p^5 \\
 &= \Pr_{X \leftarrow \mathcal{D}}[Y = F(X); X' = A(Y) : X' \in F^{-1}(Y)] \\
 &\quad - \Pr_{X \leftarrow \mathcal{D}}[Y = F(X); X' = A(Y) : X' \in F^{-1}(Y) \wedge \text{Eq}] - \hat{q}^2/2S_p^5 \\
 &> \varepsilon + 2^{-S_p} + 1/2 \\
 &\quad - \Pr[X \leftarrow \mathcal{D}; PK \parallel \vec{y} \parallel \vec{x} \parallel 1 = F(X); \\
 &\quad \quad PK' \parallel \vec{y}' \parallel M' \parallel \sigma' = A(PK \parallel \vec{y} \parallel \vec{x} \parallel 1) : M' = M] \\
 &= \varepsilon + 2^{-S_p} + 1/2 \\
 &\quad - \sum_{\vec{z}} \Pr[X \leftarrow \mathcal{D}; PK \parallel \vec{y} \parallel \vec{x} \parallel 1 = F(X); \\
 &\quad \quad PK' \parallel \vec{y}' \parallel M' \parallel \sigma' = A(PK \parallel \vec{y} \parallel \vec{x} \parallel 1) : M' = M \wedge \vec{x} = \vec{z}],
 \end{aligned}$$

where the sum is over \vec{z} consisting of \hat{q} distinct t -bit strings. Substituting \vec{z} for \vec{x} in part of the final equation above gives

$$\begin{aligned}
 \text{Succ}_B^* &\geq \varepsilon + 2^{-S_p} + 1/2 \\
 &\quad - \sum_{\vec{z}} \Pr[X \leftarrow \mathcal{D}; PK \parallel \vec{y} \parallel \vec{x} \parallel 1 = F(X); \\
 &\quad \quad PK' \parallel \vec{y}' \parallel M' \parallel \sigma' = A(PK \parallel \vec{y} \parallel \vec{z} \parallel 1) : M' = M \wedge \vec{x} = \vec{z}] \\
 &\geq \varepsilon + 2^{-S_p} + 1/2
 \end{aligned}$$

⁸Simulating a random $\pi \in \Pi_{t,n}$ is done as expected: the oracle query $x_j \parallel b$ with the j th distinct t -prefix across the executions of **Gen**, **Sign**, and **Vrfy** is answered with $\hat{y}_j \parallel b$, and an oracle query $x \parallel b$ with a previously used t -prefix is answered in a consistent manner in the obvious way.

$$- \sum_{\vec{z}} \Pr[X \leftarrow \mathcal{D}; PK \|\vec{y}\|\vec{x}\|1 = F(X);$$

$$PK' \|\vec{y}'\|M'\|\sigma' = A(PK \|\vec{y}\|\vec{z}\|1) : M' = M].$$

In this last equation we may note that A has no information about M , which is chosen at random from $\{0, 1\}^m$ independently of PK , \vec{y} , and \vec{z} . Therefore,

$$(9) \quad \text{Succ}_B^* \geq \varepsilon + 2^{-S_p} + 1/2 - \sum_{\vec{z}} 2^{-m}.$$

Noting that there are $\binom{2^t}{q} < 2^{qt} \leq 2^{m-1}$ terms in the sum of (9), we derive the contradiction $\text{Succ}_B^* > \varepsilon + 2^{-S_p}$. \square

Upper bounds on the efficiency of signature schemes. As mentioned in the introduction, our lower bounds focus on the efficiency of signature verification. We briefly observe some upper bounds on the efficiency of verification for one-time signatures (satisfying the notion of security considered here) on m -bit messages. The Lamport scheme [34] requires m invocations of a one-way permutation to verify a signature. Instead of signing bit by bit, the scheme can be modified to sign block by block. When basing the construction on an S -hard one-way permutation, it is possible to obtain provable security using blocks of length $\Theta(\log(S/m))$; this gives a signature scheme requiring only $\Theta(m/\log(S/m))$ invocations for verification. When S is polynomial, this is essentially optimal as far as verification is concerned (although the key-generation time and public-key size are prohibitive); however, the resulting scheme does not even run in polynomial time when S is super-polynomial. An alternate approach is to include a universal one-way hash function h_s as part of the public key and to use the (basic) Lamport scheme to sign $h_s(M)$. Verification now requires evaluation of h_s followed by a verification in the underlying Lamport scheme. Since h_s can be used to compress an arbitrary-length message to an n -bit string (when using an S -hard permutation on n bits) [36], we obtain a verification complexity of $\Theta(n + (m - n)/\log S)$ when $m \geq n$.

Acknowledgments. We thank the anonymous referee and Oded Goldreich for corrections and numerous comments which greatly improved the presentation.

REFERENCES

- [1] B. BARAK, *How to go beyond the black-box simulation barrier*, in 42nd IEEE Symposium on Foundations of Computer Science, IEEE, Piscataway, NJ, 2001, pp. 106–115.
- [2] B. BARAK, *Constant-round coin-tossing with a man in the middle, or realizing the shared random string model*, in 43rd IEEE Symposium on Foundations of Computer Science, IEEE, Piscataway, NJ, 2002, pp. 345–355.
- [3] D. BEAVER, *Correlated pseudorandomness and the complexity of private computations*, in 28th ACM Symposium on Theory of Computing, ACM, New York, 1996, pp. 479–488.
- [4] M. BELLARE AND O. GOLDREICH, *On defining proofs of knowledge*, in Advances in Cryptology—Crypto 1992, Lecture Notes in Comput. Sci. 740, Springer-Verlag, New York, 1993, pp. 390–420.
- [5] M. BELLARE AND S. GOLDWASSER, *New paradigms for digital signatures and message authentication based on non-interactive zero-knowledge proofs*, in Advances in Cryptology—Crypto 1989, Lecture Notes in Comput. Sci. 435, Springer-Verlag, New York, 1990, pp. 194–211.
- [6] M. BELLARE, S. HALEVI, A. SAHAI, AND S. VADHAN, *Many-to-one trapdoor functions and their relation to public-key cryptosystems*, in Advances in Cryptology—Crypto 1998, Lecture Notes in Comput. Sci. 1462, Springer-Verlag, New York, 1998, pp. 283–298.

- [7] M. BLUM AND S. GOLDWASSER, *An efficient probabilistic encryption scheme which hides all partial information*, in Advances in Cryptology—Crypto '84, Lecture Notes in Comput. Sci. 263, Springer-Verlag, New York, 1985, pp. 289–302.
- [8] M. BLUM AND S. MICALI, *How to generate cryptographically strong sequences of pseudo-random bits*, SIAM J. Comput., 13 (1984), pp. 850–864.
- [9] D. CATALANO, R. GENNARO, AND N. HOWGRAVE-GRAHAM, *Paillier's trapdoor function hides up to $O(n)$ bits*, J. Cryptology, 15 (2002), pp. 251–269.
- [10] Y. DESMEDI AND Y. FRANKEL, *Threshold cryptosystems*, in Advances in Cryptology—Crypto 1989, Lecture Notes in Comput. Sci. 435, Springer-Verlag, New York, 1990, pp. 307–315.
- [11] D. DOLEV, C. DWORK, AND M. NAOR, *Non-malleable cryptography*, SIAM J. Comput., 30 (2000), pp. 391–437.
- [12] U. FEIGE, A. FIAT, AND A. SHAMIR, *Zero-knowledge proofs of identity*, J. Cryptology, 1 (1988), pp. 77–94.
- [13] M. FISCHLIN, *On the impossibility of constructing non-interactive statistically-secret protocols from any trapdoor one-way function*, in Cryptographers' Track—RSA 2002, Lecture Notes in Comput. Sci. 2271, Springer-Verlag, New York, 2002, pp. 79–95.
- [14] R. GENNARO, Y. GERTNER, AND J. KATZ, *Lower bounds on the efficiency of encryption and digital signature schemes*, in 35th ACM Symposium on Theory of Computing, ACM, New York, 2003, pp. 417–425.
- [15] R. GENNARO AND L. TREVISAN, *Lower bounds on the efficiency of generic cryptographic constructions*, in 41st IEEE Symposium on Foundations of Computer Science, IEEE, Piscataway, NJ, 2000, pp. 305–313.
- [16] Y. GERTNER, S. KANNAN, T. MALKIN, O. REINGOLD, AND M. VISWANATHAN, *The relationship between public-key encryption and oblivious transfer*, in 41st IEEE Symposium on Foundations of Computer Science, IEEE, Piscataway, NJ, 2000, pp. 325–335.
- [17] Y. GERTNER, T. MALKIN, AND O. REINGOLD, *On the impossibility of basing trapdoor functions on trapdoor predicates*, in 42nd IEEE Symposium on Foundations of Computer Science, IEEE, Piscataway, NJ, 2001, pp. 126–135.
- [18] O. GOLDREICH, *Foundations of Cryptography, Vol. 1: Basic Tools*, Cambridge University Press, Cambridge, UK, 2001.
- [19] O. GOLDREICH, *Foundations of Cryptography, Vol. 2: Basic Applications*, Cambridge University Press, Cambridge, UK, 2004.
- [20] O. GOLDREICH, S. GOLDWASSER, AND S. MICALI, *On the cryptographic applications of random functions*, in Advances in Cryptology—Crypto '84, Lecture Notes in Comput. Sci. 263, Springer-Verlag, New York, 1985, pp. 276–288.
- [21] O. GOLDREICH, S. GOLDWASSER, AND S. MICALI, *How to construct random functions*, J. ACM, 33 (1986), pp. 792–807.
- [22] O. GOLDREICH AND L. LEVIN, *Hard-core predicates for any one-way function*, in 21st ACM Symposium on Theory of Computing, ACM, New York, 1989, pp. 25–32.
- [23] O. GOLDREICH, S. MICALI, AND A. WIGDERSON, *Proofs that yield nothing but their validity or all languages in NP have zero-knowledge proof systems*, J. ACM, 38 (1991), pp. 691–729.
- [24] O. GOLDREICH, S. MICALI, AND A. WIGDERSON, *How to play any mental game or a completeness theorem for protocols with honest majority*, in 19th ACM Symposium on Theory of Computing, ACM, New York, 1987, pp. 218–229.
- [25] S. GOLDWASSER AND S. MICALI, *Probabilistic encryption*, J. Comput. System Sci., 28 (1984), pp. 270–299.
- [26] S. GOLDWASSER, S. MICALI, AND R. L. RIVEST, *A digital signature scheme secure against adaptive chosen message attacks*, SIAM J. Comput., 17 (1988), pp. 281–308.
- [27] J. HÅSTAD, R. IMPAGLIAZZO, L. A. LEVIN, AND M. LUBY, *A pseudorandom generator from any one-way function*, SIAM J. Comput., 28 (1999), pp. 1364–1396.
- [28] J. HÅSTAD, A. SCHRIFT, AND A. SHAMIR, *The discrete logarithm modulo a composite hides $O(n)$ bits*, J. Comput. System Sci., 47 (1993), pp. 376–404.
- [29] R. IMPAGLIAZZO, *Very Strong One-Way Functions and Pseudo-random Generators Exist Relative to a Random Oracle*, manuscript, 1996.
- [30] R. IMPAGLIAZZO AND M. LUBY, *One-way functions are essential for complexity-based cryptography*, in 30th IEEE Symposium on Foundations of Computer Science, IEEE, Piscataway, NJ, 1989, pp. 230–235.
- [31] R. IMPAGLIAZZO AND S. RUDICH, *Limits on the provable consequences of one-way permutations*, in 21st ACM Symposium on Theory of Computing, ACM, New York, 1989, pp. 44–61.
- [32] J. KAHN, M. E. SAKS, AND C. D. SMYTH, *A dual version of Reimer's inequality and a proof of Rudich's conjecture*, 15th IEEE Conference on Computational Complexity, IEEE, 2000, pp. 98–103.

- [33] J. H. KIM, D. R. SIMON, AND P. TETALI, *Limits on the efficiency of one-way permutation-based hash functions*, in 40th IEEE Symposium on Foundations of Computer Science, IEEE, Piscataway, NJ, 1999, pp. 535–542.
- [34] L. LAMPORT, *Constructing Digital Signatures from a One-Way Function*, Technical report CSL-98, SRI International, Menlo Park, CA, 1979.
- [35] M. NAOR, *Bit commitment using pseudorandom generators*, J. Cryptology, 4 (1991), pp. 151–158.
- [36] M. NAOR AND M. YUNG, *Universal one-way hash functions and their cryptographic applications*, in 21st ACM Symposium on Theory of Computing, ACM, New York, 1989, pp. 33–43.
- [37] O. REINGOLD, L. TREVISAN, AND S. VADHAN, *Notions of reducibility between cryptographic primitives*, 1st Theory of Cryptography Conference, Lecture Notes in Comput. Sci. 2951, Springer-Verlag, New York, 2004, pp. 1–20.
- [38] J. ROMPEL, *One-way functions are necessary and sufficient for secure signatures*, in 22nd ACM Symposium on Theory of Computing, ACM, New York, 1990, pp. 387–394.
- [39] S. RUDICH, *Limits on the Provable Consequences of One-Way Functions*, Ph.D. thesis, University of California at Berkeley, Berkeley, CA, 1988.
- [40] S. RUDICH, *The use of interaction in public cryptosystems*, in Advances in Cryptology—Crypto 1991, Lecture Notes in Comput. Sci. 576, Springer-Verlag, New York, 1992, pp. 242–251.
- [41] D.R. SIMON, *Finding collisions on a one-way street: Can secure hash functions be based on general assumptions?*, in Advances in Cryptology—Eurocrypt 1998, Lecture Notes Comput. Sci. 1403, Springer-Verlag, New York, 1998, pp. 334–345.
- [42] A. C.-C. YAO, *Theory and application of trapdoor functions*, in 23rd IEEE Symposium on Foundations of Computer Science, IEEE, Piscataway, NJ, 1982, pp. 80–91.
- [43] A. C.-C. YAO, *How to generate and exchange secrets*, in 27th IEEE Symposium on Foundations of Computer Science, IEEE, Piscataway, NJ, 1986, pp. 162–167.

APPROXIMATING k -NODE CONNECTED SUBGRAPHS VIA CRITICAL GRAPHS*

GUY KORTSARZ[†] AND ZEEV NUTOV[‡]

Abstract. We present two new approximation algorithms for the problem of finding a k -node connected spanning subgraph (directed or undirected) of minimum cost. The best known approximation guarantees for this problem were $O(\min\{k, \frac{n}{\sqrt{n-k}}\})$ for both directed and undirected graphs, and $O(\ln k)$ for undirected graphs with $n \geq 6k^2$, where n is the number of nodes in the input graph. Our first algorithm has approximation ratio $O(\frac{n}{n-k} \ln^2 k)$, which is $O(\ln^2 k)$ except for very large values of k , namely, $k = n - o(n)$. This algorithm is based on a new result on ℓ -connected p -critical graphs, which is of independent interest in the context of graph theory. Our second algorithm uses the primal-dual method and has approximation ratio $O(\sqrt{n} \ln k)$ for all values of n, k . Combining these two gives an algorithm with approximation ratio $O(\ln k \cdot \min\{\sqrt{k}, \frac{n}{n-k} \ln k\})$, which asymptotically improves the best known approximation guarantee for directed graphs for all values of n, k , and for undirected graphs for $k > \sqrt{n}/6$. Moreover, this is the first algorithm that has an approximation guarantee better than $\Theta(k)$ for all values of n, k . Our approximation ratio also provides an upper bound on the integrality gap of the standard LP-relaxation.

Key words. connectivity, approximation, graphs, network design

AMS subject classification. 68W25

DOI. 10.1137/S0097539703435753

1. Introduction and preliminaries. A basic problem in network design is, given a graph, to find its minimum cost k -connected spanning subgraph; a graph is k -(*node*) *connected* if it is simple and there are at least k internally disjoint paths from every node to any other node. This problem is NP-hard for undirected graphs with $k = 2$ and for directed graphs with $k = 1$. The best known approximation guarantees for this problem were $O(\min\{k, \frac{n}{\sqrt{n-k}}\})$ for both directed and undirected graphs [16, 4], and $O(\ln k)$ for undirected graphs with $n \geq 6k^2$ [4], where n is the number of nodes in the input graph.¹ Better approximation guarantees are known for restricted edge costs, as follows. For metric costs, they are $2 + 2(k-1)/n$ for undirected graphs [17] (for a slight improvement to $2 + (k-1)/n$; see [16]) and $2 + k/n$ for directed graphs [16]. For uniform costs there is a $(1 + 1/k)$ -approximation algorithm for both directed and undirected graphs [3]. The case when the input graph is complete and the cost of every edge is in $\{0, 1\}$ (the so-called “vertex-connectivity augmentation problem”) is polynomially solvable for directed graphs [7]; polynomial algorithms that compute a near optimal solution for undirected graphs are given in [11, 13]. But in this paper we consider only the case of general costs.

The main result of this paper is the following theorem.

THEOREM 1.1. *There exists an algorithm for the minimum cost k -connected subgraph problem with approximation ratio $O(\ln k \cdot \min\{\sqrt{k}, \frac{n}{n-k} \ln k\})$ and running time $O(k^2 nm^2)$.*

*Received by the editors October 1, 2003; accepted for publication (in revised form) April 18, 2005; published electronically October 3, 2005.

<http://www.siam.org/journals/sicomp/35-1/43575.html>

[†]Rutgers University, Camden, NJ 08102 (guyk@camden.rutgers.edu).

[‡]Department of Computer Science, The Open University of Israel, Ramat-Aviv 61392, Israel (nutov@openu.ac.il).

¹For undirected graphs, Ravi and Williamson [28] claimed an $O(\ln k)$ -approximation algorithm, but the proof was found to contain an error; see [29].

This gives the first algorithm that has an approximation guarantee better than $\Theta(k)$ for all values of n, k and that improves the previously best known approximation guarantees for directed graphs for all values of n, k and for undirected graphs for $k > \sqrt{n/6}$. Note that our approximation ratio is $O(\ln^2 k)$ except for very large values of k (namely, $k = n - o(n)$). In particular, for instances with $n > kc$, where $c > 1$ is a fixed constant, the approximation ratio is $O(\ln^2 k)$. For example, the previously best known approximation ratios for $k = \sqrt{n}$ and $k = n/2$ were $O(k)$ and $O(\sqrt{k})$, respectively; our approximation ratio for both these cases is $O(\ln^2 k)$. For $k = n - o(n)$ the improvement is from $O(k)$ to $O(\sqrt{k} \ln k)$. Our algorithm is combinatorial and runs significantly faster than the $O(\frac{n}{\sqrt{n-k}})$ -approximation algorithm of [4], which solves linear programs.

Remark. A generalization of the minimum cost k -connected subgraph problem is the *survivable network design problem (SNDP)*, which is to find a cheapest spanning subgraph such that, for every pair (u, v) of nodes, there are at least k_{uv} pairwise internally disjoint paths from u to v . It is interesting to compare Theorem 1.1 with results in [15, 27], which show that the SNDP with $k_{uv} \in \{0, k\}$, $k = \Theta(n)$, and costs in $0, 1$, is unlikely to have a polylogarithmic approximation guarantee. On the other hand, Jain [14] showed that the version of the SNDP in which the paths are only required to be pairwise *edge* disjoint admits a 2-approximation algorithm.

Our $O(\frac{n}{n-k} \ln^2 k)$ -approximation algorithm is based on a new result on ℓ -connected p -critical graphs, which is of independent interest in graph theory. Namely, we will prove that any ℓ -connected graph (directed or not) on n nodes has a subset U of nodes with $|U| = O(\frac{n}{n-\ell} \ln \ell)$ such that no node-cut of cardinality ℓ contains U ; we call such a U an ℓ -cover (since U covers the complements of node-cuts of cardinality ℓ) and denote by $\tau_\ell(G)$ the minimum cardinality of an ℓ -cover in G . An ℓ -connected graph G is p -critical if $p < \tau_\ell(G)$ (this definition is shown to be equivalent to the one used in the papers on the topic; see section 1.2). For undirected graphs, our result partly bridges the gap between two main bounds: the obvious fact that $\tau_\ell(G) \leq \ell + 1$ and a result of Mader [21], which states that $\tau_\ell(G) \leq 3$ for $n \geq 6\ell^2$. Other previous bounds were for undirected graphs only, and of the type $\tau_\ell(G) = \Theta(\ell)$ (see, e.g., [18]), or of the type $\tau_\ell(G) = \Theta(1)$ for $n = \Omega(\ell^2)$ (see, e.g., [22]). Our result gives the first nontrivial bound in the intermediate range for undirected graphs and, overall, the first nontrivial bound for directed graphs. Moreover, our proof provides a polynomial algorithm that computes an ℓ -cover within the stated bound.

Throughout the paper, let $\mathcal{G} = (V, \mathcal{E})$ denote the input graph with nonnegative costs on the edges; n denotes the number of nodes in \mathcal{G} , and m denotes the number of edges in \mathcal{G} . Unless stated otherwise, “graph” stands for both a directed and an undirected graph.

This paper is organized as follows. In the rest of this section, we introduce a standard LP-relaxation to the minimum cost k -connected subgraph problem and state some simple facts about ℓ -outconnected graphs and p -critical graphs. In sections 2 and 3 we give our algorithm for the minimum cost k -connected subgraph problem: in section 2 we establish existence of an ℓ -cover of size $O(\frac{n}{n-\ell} \ln \ell)$ and show how to compute it, which implies an $O(\frac{n}{n-k} \ln^2 k)$ -approximation algorithm, and in section 3 we present our primal-dual $O(\sqrt{n} \ln k)$ -approximation algorithm.

1.1. LP-relaxation and ℓ -outconnected graphs. For an edge set or a graph J on a node set V and $S, T \subseteq V$ let $\delta_J(S, T)$ denote the set of edges in J going from S to T . By Menger’s theorem, there are k internally disjoint paths from a node s to a node t in a graph $G = (V, E)$ if and only if $|\delta_E(S, T)| \geq k - (n - |S \cup T|)$ for all disjoint

$S, T \subset V$ with $s \in S$ and $t \in T$. We will compare the cost of our solutions to the optima opt_k of the following LP-relaxation for the minimum cost k -node connected spanning subgraph introduced in [7] and used in [4]:

$$\begin{aligned}
 opt_k = \min \quad & \sum_{e \in \mathcal{E}} c_e x_e \\
 \text{s.t.} \quad & \sum_{e \in \delta_{\mathcal{E}}(S, T)} x_e \geq k - (n - |S \cup T|) \quad \forall \emptyset \neq S, T \subset V, S \cap T = \emptyset, \\
 & 0 \leq x_e \leq 1 \quad \forall e \in \mathcal{E}.
 \end{aligned}$$

A graph is ℓ -outconnected from a node r if it contains ℓ internally disjoint paths from r to any other node; a graph is ℓ -inconnected to r if its reverse graph is ℓ -outconnected from r (for undirected graphs these two concepts have the same meaning). Frank and Tardos [8] showed that for directed graphs, the problem of finding an ℓ -outconnected spanning subgraph of minimum cost is solvable in polynomial time; a faster algorithm with time complexity $O(\ell^2 n^2 m) = O(m^3)$ is due to Gabow [9] (observe that $n\ell = O(m)$ in an ℓ -outconnected graph).

Let $G = (V, E)$ be an ℓ -connected spanning subgraph of cost zero of a directed graph \mathcal{G} , and suppose that G has a subset U of nodes such that no node-cut of cardinality ℓ contains all of them. Then using the algorithm of [8] it is easy to get a $2|U|$ -approximation algorithm for the problem of augmenting G to be $(\ell+1)$ -connected by adding an edge set of minimum cost as follows: for each node $r \in U$ we compute an $(\ell+1)$ -outconnected spanning subgraph from r and an $(\ell+1)$ -inconnected spanning subgraph to r and take the union of these $2|U|$ subgraphs. In fact, the following lemma, which can be easily deduced from [5, Theorem 7] (see, e.g., [4]) implies that the augmenting edge set produced has cost at most $\frac{2|U|}{k-\ell} opt_k$.

LEMMA 1.2. *Let G be an ℓ -outconnected from r subgraph of cost zero of a directed graph \mathcal{G} , and for an integer p let α^p be the minimum cost of an $(\ell+p)$ -outconnected spanning subgraph of \mathcal{G} . Then $\alpha^1 \leq \alpha^p/p$. In particular, for $\ell < k$ the minimum cost of an $(\ell+1)$ -outconnected spanning subgraph of \mathcal{G} is at most $\frac{1}{k-\ell} opt_k$.*

Khuller and Raghavachari [17] observed that the algorithm of [8] implies a 2-approximation algorithm for the problem of finding an optimal ℓ -outconnected subgraph of an undirected graph, as follows. First, replace every undirected edge e of \mathcal{G} with the two antiparallel directed edges having the same ends and being of the same cost as e . Then compute an optimal ℓ -outconnected subdigraph from r and output its underlying (undirected) simple graph. Several papers used this observation for designing approximation algorithms for node connectivity problems; see, e.g., [1, 2, 4, 16].

1.2. p -critical graphs and k -connected subgraphs. Let $\kappa(G)$ denote the connectivity of G , that is, the maximum integer ℓ for which G is ℓ -connected. A (directed or undirected) graph $G = (V, E)$ is p -critical if $\kappa(G - U) = \kappa(G) - |U|$ for any $U \subset V$ with $|U| \leq p$. One can characterize p -critical graphs in terms of covers of set families, as follows. Let $G = (V, E)$ be an ℓ -connected graph. Let $X^* = X_G^* = \{v \in V - X : \delta_G(X, v) = \emptyset\}$ denote the “node complement” of X in G . We say that $X \subset V$ is an ℓ -fragment if $X^* \neq \emptyset$ and $|V - (X \cup X^*)| = \ell$. It is well known that if G is ℓ -connected, then $|V| \geq \ell + 1$, and if $|V| = \ell + 1$, then G must be a complete graph. Note that Menger’s theorem implies the following well-known statement.

PROPOSITION 1.3. *An ℓ -connected graph G (on at least $\ell + 2$ nodes) is $(\ell + 1)$ -connected if and only if G has no ℓ -fragments.*

Given a family \mathcal{F} of subsets of a groundset V we say that $U \subseteq V$ covers \mathcal{F} if U intersects every set in \mathcal{F} . Let $\mathcal{F}_\ell(G)$ be the family of all ℓ -fragments of G . We say that $U \subseteq V$ is an ℓ -cover of G if U covers $\{X \cup X^* : X \in \mathcal{F}_\ell(G)\}$; let $\tau_\ell(G)$ denote the minimum cardinality of an ℓ -cover of G . From Proposition 1.3 we have the following.

PROPOSITION 1.4. *Let $G = (V, E)$ be a graph with $\kappa(G) = \ell$ and $|V| \geq \ell + 2$. Then*

- (i) *G is p -critical if and only if for any $U \subseteq V$ with $|U| \leq p$ there exists an ℓ -fragment X with $U \cap (X \cup X^*) = \emptyset$. Thus if G is p -critical, then G is p' -critical for any $p' \leq p$, and $\tau_\ell(G) - 1$ is the maximum p for which G is p -critical.*
- (ii) *U is an ℓ -cover of G if and only if there exists an edge set F incident to U (that is, every edge in F has at least one endpoint in U) such that $G + F$ is $(\ell + 1)$ -connected.*

Combining Proposition 1.4(ii) with Lemma 1.2 and the discussion before it, we get the following statement, which was implicitly proved in [4] for undirected graphs.

PROPOSITION 1.5 (see [4]). *Suppose there is a polynomial algorithm that finds in any ℓ -connected graph G on n nodes an ℓ -cover of G of size at most $t(\ell, n)$. Then there exists a polynomial algorithm that, for instances of the minimum k -connected subgraph problem on n nodes, finds a feasible solution of cost at most $opt_k \cdot 2 \sum_{\ell=0}^{k-1} \frac{t(\ell, n)}{k-\ell} = opt_k \cdot O(\ln k \cdot \max_{0 \leq \ell \leq k-1} t(\ell, n))$.*

For undirected graphs with $n \geq 6k^2$, Cheriyan, Vempala, and Vetta [4] gave a $6H(k)$ -approximation algorithm for the undirected minimum cost k -connected subgraph problem combining Proposition 1.5 with the following theorem due to Mader.

THEOREM 1.6 (see [21]). *Any undirected 3-critical graph G has less than $6\kappa(G)^2$ nodes.*

In a recent paper, Mader [22] improved his bound for 3-critical graphs to $n \leq \kappa(G)(2\kappa(G) - 1)$; hence via Proposition 1.5 the $6H(k)$ -approximation algorithm of [4] is valid for $n \geq k(2k - 1)$ as well.

On the other hand, it is easy to see that there are no $\kappa(G)$ -critical noncomplete graphs. But for undirected graphs, a stronger result was conjectured in [26] and answered by Su as follows.

THEOREM 1.7 (see [30]). *If a noncomplete graph G is p -critical, then $p \leq \lfloor \kappa(G)/2 \rfloor$.*

For a survey on p -critical graphs see [24, 25]; for some recent results see [22, 23] and [18].

2. Computing logarithmic covers. Note that in terms of covers of set families Theorem 1.6 states that $\tau_\ell(G) \leq 3$ for any undirected graph G with $\kappa(G) = \ell$ and $n \geq 6\ell^2$, and Theorem 1.7 states that $\tau_\ell(G) \leq \lfloor \ell/2 \rfloor + 1$ (if $n \geq \ell + 2$). Our result on p -critical graphs partly bridges the gap between these two bounds, and also gives the first nontrivial bound on $\tau_\ell(G)$ for directed graphs. Let $\theta = \theta(\ell, n) = \frac{2n}{n+\ell}$.

THEOREM 2.1. *There exists a polynomial algorithm that, given an ℓ -connected graph G on $n \geq \ell + 2$ nodes, finds an ℓ -cover of G of size at most*

$$t(\ell, n) = 2 + \frac{3n}{n-\ell} + \frac{1}{\ln \theta} \ln \frac{1}{2} \left(\ell - 1 - \frac{\ell^2}{n} \right) = O \left(\frac{n}{n-\ell} \ln \ell \right)$$

if G is undirected and of size at most $2t(\ell, n)$ if G is directed.

Combining with Proposition 1.5 we get the following.

THEOREM 2.2. *For the minimum cost k -connected subgraph problem there exists a polynomial algorithm that finds a feasible solution of cost at most $opt_k \cdot O(\frac{n}{n-k} \ln^2 k)$.*

Remark. Note that $\tau_\ell(G)$ is the minimum cardinality of a cover (transversal) of the $(n - \ell)$ -uniform hypergraph $\{X \cup X^* : X \in \mathcal{F}_\ell(G)\}$. Several general bounds on covers of uniform hypergraphs are known; see, e.g., [10]. But, as far as we can see, none of them implies the bound given in Theorem 2.1.

We need several definitions and simple facts to prove Theorem 2.1. In the rest of this section, let ℓ be a fixed integer, and let G be a graph with $\kappa(G) \geq \ell$. An ℓ -fragment X of G is *small* if $|X| \leq |X^*|$, that is, if $|X| \leq \lfloor (n - \ell)/2 \rfloor$. Note that by Proposition 1.3, G is $(\ell + 1)$ -connected if and only if G (and the reverse graph of G , if G is directed) has no small ℓ -fragments. Let $\mathcal{S}_\ell(G)$ denote the family of all small ℓ -fragments of G . The following lemma is well known; see, e.g., [11, Lemma 1.2], where it was stated for undirected graphs.

LEMMA 2.3. *Let X, Y be two intersecting ℓ -fragments in an ℓ -connected (directed or undirected) graph G on n nodes. If $n - |X \cup Y| \geq \ell$, then $X \cap Y$ is an ℓ -fragment, and if a strict inequality holds, then $X \cup Y$ also is an ℓ -fragment. In particular, the intersection of two intersecting small ℓ -fragments is also a small ℓ -fragment.*

A *core* of G is an inclusion minimal small ℓ -fragment. By Lemma 2.3 the cores of G are pairwise disjoint and $\nu(G) = \nu_\ell(G)$ denote their number; note that if $\kappa(G) > \ell$, then $\nu_\ell(G) = 0$. For a core C_i of G , let A_i be the union of all small ℓ -fragments that contain a unique core, which is C_i . Let $\mathcal{A}_\ell(G) = \{A_1, \dots, A_{\nu(G)}\}$. The properties of the sets in $\mathcal{A}_\ell(G)$ that we use are summarized in the following statement.

COROLLARY 2.4. *The sets in $\mathcal{A}_\ell(G)$ are pairwise disjoint. Moreover, for every $A \in \mathcal{A}_\ell(G)$ it holds that either A is an ℓ -fragment or $|A| \geq n - \ell$ (and $A^* = \emptyset$); thus $|A \cup A^*| \geq n - \ell$.*

Proof. Suppose to the contrary that A_i and A_j intersect for some $1 \leq i \neq j \leq \nu(G)$. Then, by the definition of A_i, A_j , there are two small ℓ -fragments D_i, D_j such that D_i contains a unique core which is C_i , D_j contains a unique core which is C_j , and D_i, D_j intersect. By Lemma 2.3 $D_i \cap D_j$ is a small ℓ -fragment, and thus contains a core C . This implies that D_i contains the two cores C_i and C , which gives a contradiction.

To prove the second statement, let us fix some set $A \in \mathcal{A}_\ell(G)$. Since the sets in $\mathcal{A}_\ell(G)$ are disjoint, A contains a unique core, say C . Consider the family \mathcal{D} of all small ℓ -fragments that contain a unique core, which is C , so A is the union of the sets in \mathcal{D} . If $n - |A| \leq \ell$, then clearly $|A \cup A^*| \geq |A| \geq n - \ell$ (in fact, in this case $A^* = \emptyset$, and thus $|A \cup A^*| = |A|$). Otherwise, $n - |A| \geq \ell + 1$; then by Lemma 2.3, the union of the sets in \mathcal{D} is an ℓ -fragment, and thus $|A \cup A^*| = n - \ell$. In both cases, the statement is valid. \square

Note that Corollary 2.4 does *not* imply that the sets in $\mathcal{A}_\ell(G)$ are small or that they are ℓ -fragments; it might be that A is large and that $A^* = \emptyset$ for some $A \in \mathcal{A}_\ell(G)$, but in any case, $|A \cup A^*| \geq n - \ell$ holds.

LEMMA 2.5. *Let \mathcal{A} be a family of sets on a groundset V such that $|A| \geq n - \ell$ holds for every $A \in \mathcal{A}$, where $n = |V|$ and ℓ is an integer. Then there exists an element $r \in V$ that covers (that is, intersects) at least $(1 - \frac{\ell}{n})|\mathcal{A}|$ sets in \mathcal{A} .*

Proof. For $r \in V$, let $\mathcal{A}_r = \{A \in \mathcal{A} : r \in A\}$ be the sets in \mathcal{A} covered by r . The claim follows since we have

$$\sum_{r \in V} |\mathcal{A}_r| = \sum_{A \in \mathcal{A}} |A| \geq |\mathcal{A}|(n - \ell). \quad \square$$

For $r \in V$ let $F_r = \{vr : v \in V - r\}$, and let $G + F_r$ be the graph obtained by

adding an edge from every node $v \in V$ to r if such does not exist in G . We say that $r \in V$ *outercovers* $A \in \mathcal{A}_\ell(G)$ if $r \in A^*$.

LEMMA 2.6. *Let C_i be a core of an ℓ -connected graph G . If r outercovers A_i , then any small ℓ -fragment X of $G + F_r$ that contains C_i must contain a core of G distinct from C_i .*

Proof. Let X be a small ℓ -fragment of $G + F_r$ that contains C_i . Assume by contradiction that this is the unique core of G that X contains. Note that X is a small ℓ -fragment of G . Since A_i is defined as the union of all small ℓ -fragments of G containing C_i as their unique core, we obtain that $X \subseteq A_i$. This gives a contradiction, since then $r \in A_i^* \subseteq X^*$, which implies that X cannot be an ℓ -fragment of $G + F_r$. \square

LEMMA 2.7. *Let r be a node that outercovers q sets in $\mathcal{A}_\ell(G)$. Then $\nu_\ell(G + F_r) \leq \nu_\ell(G) - q/2$.*

Proof. If $\kappa(G + F_r) > \ell$, then $\nu_\ell(G + F_r) = 0$ and the statement is obvious, so assume that $\kappa(G + F_r) = \ell$. By Lemma 2.3, the cores of $G + F_r$ are pairwise disjoint. Clearly, every core of $G + F_r$ is a small ℓ -fragment of G , and thus contains at least one core of G . Let t be the number of cores of $G + F_r$ containing exactly one core of G . By Lemma 2.6, any core C of $G + F_r$ that contains some core C_i of G with $r \in A_i^*$ must contain another core of G distinct from C_i , so such C contains at least two cores of G . Thus $t \leq \nu_\ell(G) - q$. From this we get that $\nu_\ell(G + F_r) \leq t + (\nu_\ell(G) - t)/2 \leq \nu_\ell(G) - q/2$, as required. \square

Since the sets in $\mathcal{A}_\ell(G)$ are pairwise disjoint, a node can belong to at most one of them. Thus, if $\mathcal{A}' \subseteq \mathcal{A}_\ell(G)$ and r covers $\{A \cup A^* : A \in \mathcal{A}'\}$, then there is at most one set $A' \in \mathcal{A}'$ such that $r \in A'$; for any other $A \in \mathcal{A} - A'$ we must have $r \in A^*$; hence r outercovers at least $|\mathcal{A}'| - 1$ sets in \mathcal{A}' . Combining this with Corollary 2.4 and Lemmas 2.5 and 2.7 we get the following.

COROLLARY 2.8. *Any ℓ -connected graph G has a node r that outercovers at least $\nu_\ell(G)(1 - \ell/n) - 1$ sets in $\mathcal{A}_\ell(G)$, and $\nu_\ell(G + F_r) \leq \frac{n+\ell}{2n}\nu_\ell(G) + 1/2$.*

Let us apply the following algorithm on an ℓ -connected graph G starting with $U = \emptyset$.

While $\nu_\ell(G) > 0$ do:

1. Find a node r for which $\nu_\ell(G + F_r)$ is minimal;
2. $U \leftarrow U + r, G \leftarrow G + F_r$;

End While

Output U .

By Proposition 1.4(ii), at the end of the algorithm U is an ℓ -cover; let us estimate its size. Let t_j be the number of cores in G after j iterations of the main loop, and set $\theta = \frac{2n}{n+\ell}$ and $\alpha = 1/\theta$. Corollary 2.8 gives the recursive bound

$$t_{j+1} \leq \alpha t_j + 1/2.$$

We will prove later that $t_1 \leq \ell$ (see Corollary 2.11 below) which implies

$$\begin{aligned} t_j &\leq \alpha^{j-1}\ell + \frac{1}{2}(1 + \alpha + \dots + \alpha^{j-2}) = \alpha^{j-1}\ell + \frac{1 - \alpha^{j-1}}{2(1 - \alpha)} \\ &= \alpha^{j-1} \left(\ell - \frac{1}{2(1 - \alpha)} \right) + \frac{1}{2(1 - \alpha)} = \frac{1}{\theta^{j-1}} \left(\ell - \frac{n}{n - \ell} \right) + \frac{n}{n - \ell}. \end{aligned}$$

The inequality can be easily proved by induction on j . Let $\beta = 3n/(n - \ell)$.

CLAIM 2.9.

$$t_j \leq \beta \quad \text{for} \quad j \geq j(\beta) \equiv \frac{1}{\ln \theta} \ln \frac{1}{2} \left(\ell - 1 - \frac{\ell^2}{n} \right) + 1.$$

Proof. We solve for j the inequality

$$\frac{1}{\theta^{j-1}} \left(\ell - \frac{n}{n-\ell} \right) + \frac{n}{n-\ell} \leq \frac{3n}{n-\ell} = \beta.$$

That is,

$$\theta^{j-1} \geq \frac{\ell(n-\ell) - n}{2n} = \frac{1}{2} \left(\ell - 1 - \frac{\ell^2}{n} \right).$$

The claim follows by taking logarithm base θ from both sides and then changing the logarithm base as follows:

$$j - 1 \geq \log_{\theta} \frac{1}{2} \left(\ell - 1 - \frac{\ell^2}{n} \right) = \frac{1}{\ln \theta} \ln \frac{1}{2} \left(\ell - 1 - \frac{\ell^2}{n} \right). \quad \square$$

On the other hand, if $t_j > 0$, then $t_{j+1} \leq t_j - 1$ since $\nu_{\ell}(G + F_r) \leq \nu_{\ell}(G) - 1$ for any node r belonging to a core C_i of G (indeed, every core of $G + F_r$ must contain some core of G , but cannot contain C_i). Thus the number of iterations in the algorithm (which is equal to the size of the cover found) is bounded by $\lceil j(\beta) \rceil + \lfloor \beta \rfloor \leq t(\ell, n)$. This proves Theorem 2.1 for undirected graphs. In the case of a directed graph G , at the end of the algorithm, G has no small ℓ -fragments, but G may not be $(\ell + 1)$ -connected since the reverse graph of G might have small ℓ -fragments. Thus we apply the above procedure twice, that is, on G and on the reverse graph of G , and take the union of the resulting two node sets.

Let us now show that $t_1 \leq \ell$ and discuss some consequences of our approach. The following statement is obvious.

LEMMA 2.10. *Let r be a node of a (directed or undirected) noncomplete graph $G = (V, E)$ with $\kappa(G) = \ell$, and let $N_r = \{v \in V : vr \in E\}$ be the nodes in G having r as their neighbor. Then N_r covers all ℓ -fragments of $G + F_r$.*

An ℓ -connected graph J is *minimally ℓ -connected* if $J - e$ is not ℓ -connected for every edge e of J . Mader [19, 20] showed that any minimally ℓ -connected graph J on n nodes has at least $\frac{(\ell-1)n+2}{2^{\ell-1}}$ nodes of degree (indegree, if J is directed) ℓ . Since $\mathcal{F}_{\ell}(G) \subseteq \mathcal{F}_{\ell}(J)$ for any ℓ -connected spanning subgraph J of an ℓ -connected graph G , Lemma 2.10 implies the following corollary, which also proves that $t_1 \leq \ell$.

COROLLARY 2.11. *Let $G = (V, E)$ be an ℓ -connected graph. Then there is $R \subseteq V$ with $|R| \geq \frac{(\ell-1)n+2}{2^{\ell-1}}$ such that for any $r \in R$ the following holds: r and at most ℓ nodes having r as their neighbor cover all ℓ -fragments of G and, in particular, $\nu_{\ell}(G + F_r) \leq \ell$.*

We note that for a directed graph G , Corollary 2.11 implies only the trivial bound $\tau_{\ell}(G) \leq \ell + 1$; however, for undirected G the following theorem provides an easy proof of Theorem 1.7 and is similar to the proof given by Jordán in [12]; recall that in terms of covers, Theorem 1.7 states that $\tau_{\ell}(G) \leq \lfloor \ell/2 \rfloor + 1$.

THEOREM 2.12. *Let G be an undirected ℓ -connected graph and let W be a cover of $\mathcal{F}_{\ell}(G)$. Then there exists an ℓ -cover $U \subseteq W$ of size at most $\lfloor |W|/2 \rfloor$.*

Proof. In [19], Mader implicitly proved (see, e.g., [11] and [16, Corollary 2.2]) that if W covers all the ℓ -fragments of an undirected ℓ -connected graph G , then there

exists a forest F on W such that $G + F$ is $(\ell + 1)$ -connected. Since F is a forest, there exists $U \subseteq W$ such that $|U| \leq \lfloor |W|/2 \rfloor$ and every edge in F is incident to a node in U . Thus, by Proposition 1.4(ii), U is a cover of $\mathcal{F}_\ell(G)$ as required. \square

Let us now analyze the time complexity of our approximation algorithm for k -connected spanning subgraphs. Using max-flow techniques, we find that an ℓ -cover as in Theorem 2.1 can be found in $O(\ell m^2)$ time, as follows. For the first iteration, we find a minimally ℓ -connected spanning subgraph J of G and choose a node s of degree (indegree, if G is directed) ℓ in J ; such J can be found in $O(\ell m^2)$ time by repeatedly checking every edge for deletion. By Lemma 2.10 the set $N = N_s$ of nodes having s as their neighbor in J covers all ℓ -fragments of $J + F_s$, and thus also of $G + F_s$. Now we set $G \leftarrow G + F_s$. We compute for every $u \in N_s$ and $v \in V$ a set of ℓ internally disjoint paths. This can be done in $O(\ell m)$ time per pair, and thus in $O(\ell^2 nm)$ total time, using the Ford–Fulkerson algorithm (the node-capacitated version) and flow decomposition. For each pair uv we check whether v is reachable from u in the corresponding residual network. If so, then the pair uv is discarded; otherwise, a minimal ℓ -fragment containing v is found, and if its size is $\leq (n - \ell)/2$, it is the minimal core containing v . At each iteration, for every $r \in V$, we can recompute the cores of $G + F_r$ in $O(\ell m)$ time. Thus each iteration can be implemented in $O(\ell nm)$ time, and since the number of iterations is at most ℓ , an ℓ -cover as in Theorem 2.1 can be found in $O(\ell^2 nm) = O(\ell m^2)$ time, as claimed.

We also need to find a minimum cost edge set to increase the outconnectivity from ℓ to $\ell + 1$ from each node in the cover found. Frank [6] showed that a generalization of this problem can be solved in $O(n^2 m)$ time, but with some care Frank’s algorithm can be implemented in $O(m^2)$ time. As the size of the cover found is $O(\ell)$, we get that the overall time complexity for increasing connectivity from ℓ to $\ell + 1$ is $O(\ell m^2)$, where $m \geq n\ell$ is the number of edges in \mathcal{G} . Consequently, the overall running time of the algorithm is $O(k^2 m^2)$.

3. A primal-dual algorithm. In this section we prove the following theorem.

THEOREM 3.1. *For the problem of making a k_0 -connected graph (directed or undirected) k -connected by adding a minimum cost edge set, there exists an approximation algorithm with approximation ratio $O(\sqrt{n}H(k - k_0)) = O(\sqrt{n} \ln k)$ and time complexity $O(km(k^2 n^2 + \sqrt{nm})) = O(n^5 m)$, where $H(j)$ denotes the j th Harmonic number.*

We start by giving an algorithm for increasing the connectivity of a *directed graph* by one. We use as a subroutine the primal-dual algorithm of Ravi and Williamson [28], which we adapt to directed graphs. Given an ℓ -connected graph G , the algorithm of [28] uses the primal-dual method to find an edge set F so that $G + F$ is $(\ell + 1)$ -connected. We use the same approach, but unlike the algorithm in [28], the primal-dual procedure terminates when we find an edge set F^+ so that $\nu_\ell(G + F^+) = \sqrt{2n}$; we will show that $c(F^+) = O(\frac{\sqrt{n}}{k-\ell})opt_k$. We then find in $G + F^+$ a node set U of size $O(\sqrt{n})$ by picking one node from every core; for every $r \in U$ we find an $(\ell + 1)$ -inconnected subgraph to r subgraph. The cost of each subgraph found is $O(\frac{1}{k-\ell})opt_k$ and its number is $|U| = O(\sqrt{n})$. Thus the cost of the edge set found during this step is also $O(\frac{\sqrt{n}}{k-\ell})opt_k$. We apply this procedure twice, that is, on G and on the reversed graph of G . Consequently, the total cost of the edge set found is $O(\frac{\sqrt{n}}{k-\ell})opt_k$.

Let $G = (V, E)$ be an ℓ -connected spanning subgraph of a directed graph $\mathcal{G} = (V, \mathcal{E})$ such that all the edges in E have cost zero, and let $I = \mathcal{E} - E$. Let $\mathcal{S} = \mathcal{S}_\ell(G)$ denote the set of small ℓ -fragments of G . For an edge set F and $S \in \mathcal{S}$, let $d_F(S) =$

$|\delta_F(S, S^*)|$ be the number of edges in F going from S to S^* . Recall that $G + F$ is $(\ell + 1)$ -connected if and only if the graph $G + F$ and its reverse graph have no small ℓ -fragments. Note that for $F \subseteq I = \mathcal{E} - E$, $G + F$ has no small ℓ -fragments if and only if $d_F(S, S^*) \geq 1$ for any $S \in \mathcal{S}$. Consider the following linear program (P) and its dual program (D), where (P) is a linear relaxation for the problem of finding a minimum cost augmenting edge set F such that $G + F$ contains no small ℓ -fragments:

$$\begin{aligned}
 \text{(P)} \quad & \min \quad \sum_{e \in I} c_e x_e \\
 \text{s.t.} \quad & \sum_{e \in \delta_I(S, S^*)} x_e \geq 1 \quad \forall S \in \mathcal{S}, \\
 & x_e \geq 0 \quad \forall e \in I.
 \end{aligned}$$

$$\begin{aligned}
 \text{(D)} \quad & \max \quad \sum_{S \in \mathcal{S}} y_S \\
 \text{s.t.} \quad & \sum_{S \in \mathcal{S}: e \in \delta_I(S, S^*)} y_S \leq c_e \quad \forall e \in I, \\
 & y_S \geq 0 \quad \forall S \in \mathcal{S}.
 \end{aligned}$$

LEMMA 3.2. *Let x be an optimal solution to (P). Then $\sum_{e \in I} c_e x_e \leq \frac{1}{k-\ell} \text{opt}_k$.*

Proof. Let x be an optimal solution to the LP-relaxation for the minimum cost k -connected spanning subgraph problem (given in section 1.1). Define $x'_e = 1$ if $e \in E$ and $x'_e = \frac{1}{k-\ell} x_e$ otherwise. Then x' is a feasible solution to (P). Since all the edges in E have zero cost, the claim follows. \square

Given a feasible solution y to (D), an edge $e \in I$ is *tight* if the corresponding inequality in the dual program (D) holds with equality. If $F^+ \subseteq I$ is a set of tight edges, then

$$(1) \quad c(F^+) = \sum_{e \in F^+} c_e = \sum_{e \in F^+} \sum_{S \in \mathcal{S}: e \in \delta(S, S^*)} y_S = \sum_{S \in \mathcal{S}} d_{F^+}(S) y_S.$$

Recall that by Lemma 2.3 the cores of G are disjoint. Let us fix the threshold $\beta = \sqrt{2n}$ and apply the following procedure.

Procedure 1. While $\nu_\ell(G) \geq \beta$, raise dual variables corresponding to cores of G uniformly until some edge $e \in I$ becomes tight, and add this edge to G .

Let \tilde{F}^+ be the set of edges added to the input graph G by Procedure 1.

LEMMA 3.3. *Let $F^+ \subseteq \tilde{F}^+$. Then*

$$c(F^+) \leq \frac{|F^+|}{\beta} \frac{\text{opt}_k}{k-\ell}.$$

Proof. Let y be the dual solution produced by Procedure 1. Since the edges in F^+ are tight, we have $c(F^+) = \sum_{S \in \mathcal{S}} d_{F^+}(S) y_S$ by (1). Let \mathcal{C}_i be the family of cores of G at iteration i , and let ε_i be the amount at which they were raised at iteration i , $i = 1, \dots, q$. Note that $y_S = \sum_{\{i: S \in \mathcal{C}_i\}} \varepsilon_i$ for any set $S \in \mathcal{S}$. Using this, together with the fact that the sets in \mathcal{C}_i are disjoint and that $|\mathcal{C}_i| \geq \beta$, we get

$$\begin{aligned}
 \sum_{S \in \mathcal{S}} d_{F^+}(S) y_S &= \sum_{i=1}^q \varepsilon_i \sum_{S \in \mathcal{C}_i} d_{F^+}(S) \leq \sum_{i=1}^q \varepsilon_i |F^+| \frac{|\mathcal{C}_i|}{\beta} = \frac{|F^+|}{\beta} \sum_{i=1}^q \varepsilon_i |\mathcal{C}_i| \\
 &= \frac{|F^+|}{\beta} \sum_{S \in \mathcal{S}} y_S \leq \frac{|F^+|}{\beta} \frac{\text{opt}_k}{k-\ell}.
 \end{aligned}$$

The first inequality follows by upper bounding 1 by $|\mathcal{C}_i|/\beta$ and noting that in $\sum_{S \in \mathcal{C}_i} d_{F^+}(S)$ every edge is counted exactly once (since the graphs we consider are directed). The last inequality follows from Lemma 3.2 and the weak duality theorem. \square

After executing Procedure 1, let C_1, \dots, C_{ν^+} be the cores of $G + \tilde{F}^+$.

Procedure 2. For $j = 1, \dots, \nu^+$, choose $r_j \in C_j$, and compute an optimal edge set \tilde{F}_j^+ such that $G + \tilde{F}_j^+$ is $(\ell + 1)$ -outconnected from r_j .

Note that by Lemma 1.2, $c(\tilde{F}_j^+) \leq \frac{1}{k-\ell} \text{opt}_k$, $j = 1, \dots, \nu^+$.

We then apply Procedures 1 and 2 on the reverse graph of $G + I$ to find appropriate edge sets \tilde{F}^- and $\tilde{F}_1^-, \dots, \tilde{F}_{\nu^-}^-$. Let \tilde{F} be the union of all the edge sets found. Then $G + \tilde{F}$ is $(\ell + 1)$ -connected. The last step in our algorithm is finding an inclusion minimal edge set $F \subseteq \tilde{F}$ such that $G + F$ is $(\ell + 1)$ -connected. Note that $|\tilde{F}|$ might be large, but the following statement shows that $|F| = O(n)$.

THEOREM 3.4 (see [20]). *Let G be an ℓ -connected directed graph, and let F be an inclusion minimal augmenting edge set such that $G + F$ is $(\ell + 1)$ -connected. Then $|F| \leq 2n - 1$.*

LEMMA 3.5. *The algorithm produces a feasible solution of cost at most $\frac{4\sqrt{2n}}{k-\ell} \text{opt}_k$.*

Proof. By Theorem 3.4, $|F| \leq 2n$. Set $F^+ = \tilde{F}^+ \cap F$, $F^- = \tilde{F}^- \cap F$, $F_j^+ = \tilde{F}_j^+ \cap F$ for $j = 1, \dots, \nu^+$, and $F_j^- = \tilde{F}_j^- \cap F$ for $j = 1, \dots, \nu^-$. Applying Lemmas 3.3 and 1.2, Theorem 3.4, and recalling that $\nu^+, \nu^- \leq \beta \leq \sqrt{2n}$, we get

$$c(F) \leq c(F^+) + c(F^-) + \sum_{j=1}^{\nu^+} c(F_j^+) + \sum_{j=1}^{\nu^-} c(F_j^-) \leq \frac{2\text{opt}_k}{k-\ell} \left(\frac{2n}{\beta} + \beta \right) \leq \frac{4\sqrt{2n}}{k-\ell} \text{opt}_k. \quad \square$$

Suppose now that the input graph \mathcal{G} contains a k_0 -connected spanning subgraph of cost zero. We can repeatedly apply the above algorithm starting with $\ell = k_0$ until $\ell = k - 1$ to compute a k -connected spanning subgraph of \mathcal{G} ; the overall cost of the subgraph found will be at most $4\sqrt{2n}H(k - k_0)\text{opt}_k = O(\sqrt{n} \ln k)\text{opt}_k$.

For undirected graphs, an $8\sqrt{2n}H(k - k_0)$ -approximation algorithm easily follows using the reduction due to Khuller and Raghavachari [17] described at the end of section 1.1.

To finish the proof of Theorem 3.1, let us discuss the implementation and the time complexity of the algorithm. As was mentioned, Procedure 1 in our algorithm is similar to the procedure used in [28], and we can adapt the implementation of [28] as well. We omit the details but note that for implementing all Procedures 1 in the algorithm, as well as finding minimal edge sets $F \subseteq \tilde{F}$ such that $G + F$ is $(\ell + 1)$ -connected, $\ell = 0, \dots, k - 1$, can be done in $O(k^3mn^2) = O(k^2nm^2)$ total time; see section 5 in [28]. Using the algorithm of [9], the overall time required for Procedure 2 implementations is $O(k^3mn^2\sqrt{n})$. Note however, that Procedure 2 requires finding a minimum cost edge set to increase the outconnectivity from ℓ to $\ell + 1$. As was mentioned, this problem can be solved in $O(m^2)$ time using Frank's algorithm [6]. Thus the total time required for Procedure 2 executions is $O(k\sqrt{nm}^2)$.

Acknowledgments. The authors thank Joseph Cheriyan and an anonymous referee for their comments.

REFERENCES

[1] V. AULETTA, Y. DINITZ, Z. NUTOV, AND D. PARENTE, *A 2-approximation algorithm for finding an optimum 3-vertex connected spanning subgraph*, J. Algorithms, 32 (1999), pp. 21–30.

- [2] J. CHERIYAN, T. JORDÁN, AND Z. NUTOV, *On rooted node-connectivity problems*, *Algorithmica*, 30 (2001), pp. 353–375.
- [3] J. CHERIYAN AND R. THURIMELLA, *Approximating minimum-size k -connected spanning subgraphs via matching*, *SIAM J. Comput.*, 30 (2000), pp. 528–560.
- [4] J. CHERIYAN, S. VEMPALA, AND A. VETTA, *An approximation algorithm for the minimum-cost k -vertex connected subgraph*, *SIAM J. Comput.*, 32 (2003), pp. 1050–1055.
- [5] A. FRANK, *Connectivity augmentation problems in network design*, in *Mathematical Programming: State of the Art*, J. R. Birge and K. G. Murty, eds., University of Michigan, Ann Arbor, 1994, pp. 34–63.
- [6] A. FRANK, *Increasing the rooted connectivity of a digraph by one*, *Math. Programming*, 84 (1999), pp. 565–576.
- [7] A. FRANK AND T. JORDÁN, *Minimal edge-coverings of pairs of sets*, *J. Combin. Theory Ser. B*, 65 (1995), pp. 73–110.
- [8] A. FRANK AND É. TARDOS, *An application of submodular flows*, *Linear Algebra Appl.*, 114/115 (1989), pp. 329–348.
- [9] H. N. GABOW, *A representation for crossing set families with application to submodular flow problems*, in *Proceedings of the 4th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, SIAM, Philadelphia, 1993, pp. 202–211.
- [10] L. GRAHAM, M. GRÖTSCHEL, AND L. LOVÁSZ, EDS., *Handbook of Combinatorics, Vol. 1*, Elsevier Science B.V., Amsterdam; MIT Press, Cambridge, MA, 1995.
- [11] T. JORDÁN, *On the optimal vertex-connectivity augmentation*, *J. Combin. Theory Ser. B*, 63 (1995), pp. 8–20.
- [12] T. JORDÁN, *On the existence of (k, ℓ) -critical graphs*, *Discrete Math.*, 179 (1998), pp. 273–275.
- [13] B. JACKSON AND T. JORDÁN, *A near optimal algorithm for vertex connectivity augmentation*, in *Algorithms and Computations, Lecture Notes in Comput. Sci. 1969*, Springer, Berlin, 2000, pp. 313–325.
- [14] K. JAIN, *A factor 2 approximation algorithm for the generalized Steiner network problem*, *Combinatorica*, 21 (2001), pp. 39–60.
- [15] G. KORTSARZ, R. KRAUTHGAMER, AND J. LEE, *Hardness of approximation for vertex-connectivity network design problems*, *SIAM J. Comput.*, 33 (2004), pp. 704–720.
- [16] G. KORTSARZ AND Z. NUTOV, *Approximating node connectivity problems via set covers*, *Algorithmica*, 37 (2003), pp. 75–92.
- [17] S. KHULLER AND B. RAGHAVACHARI, *Improved approximation algorithms for uniform connectivity problems*, *J. Algorithms*, 21 (1996), pp. 434–450.
- [18] M. KRIESELL, *Upper bounds to the number of vertices in a k -critically n -connected graph*, *Graphs Combin.*, 18 (2002), pp. 133–146.
- [19] W. MADER, *Ecken vom Grad n in minimalen n -fach zusammenhängenden Graphen*, *Arch. Math. (Basel)*, 23 (1972), pp. 219–224.
- [20] W. MADER, *Minimal n -fach zusammenhängende digraphen*, *J. Combin. Theory Ser. B*, 38 (1985), pp. 102–117.
- [21] W. MADER, *Endlichkeitsätze für k -kritische graphen*, *Math. Ann.*, 229 (1977), pp. 143–153.
- [22] W. MADER, *On k -con-critically n -connected graphs*, *J. Combin. Theory Ser. B*, 86 (2002), pp. 296–314.
- [23] W. MADER, *High connectivity keeping sets in n -connected graphs*, *Combinatorica*, 24 (2004), pp. 441–458.
- [24] W. MADER, *Connectivity and edge-connectivity in finite graphs*, in *Surveys in Combinatorics*, London Math. Soc. Lecture Notes Ser. 38, Cambridge University Press, Cambridge, UK, 1979, pp. 66–95.
- [25] W. MADER, *On k -critically n -connected graphs*, in *Progress in Graph Theory*, Academic Press, Toronto, ON, 1984, pp. 389–398.
- [26] B. MAURER AND P. SLATER, *On k -critical, n -connected graphs*, *Discrete Math.*, 20 (1988), pp. 255–262.
- [27] Z. NUTOV, *Approximating connectivity augmentation problems*, in *Proceedings of the 16th Annual ACM-SIAM Symposium on Discrete Algorithms*, SIAM, Philadelphia, 2005, pp. 176–185.
- [28] R. RAVI AND D. P. WILLIAMSON, *An approximation algorithm for minimum-cost vertex-connectivity problems*, *Algorithmica*, 18 (1997), pp. 21–43.
- [29] R. RAVI AND D. P. WILLIAMSON, *Erratum: An approximation algorithm for minimum-cost vertex-connectivity problems*, *Algorithmica*, 34 (2002), pp. 98–107.
- [30] J. SU, *Proof of Slater’s conjecture on k -critical n -connected graphs*, *Kexue Tongbao (English Ed.)*, 33 (1988), pp. 1675–1678.

A PARAMETRIZED ALGORITHM FOR MATROID BRANCH-WIDTH*

PETR HLINĚNÝ†

Abstract. Branch-width is a structural parameter very closely related to tree-width, but branch-width has an immediate generalization from graphs to matroids. We present an algorithm that, for a given matroid M of bounded branch-width t which is represented over a finite field, finds a branch decomposition of M of width at most $3t$ in cubic time. Then we show that the branch-width of M is a uniformly fixed-parameter tractable problem. Other applications include recognition of matroid properties definable in the monadic second-order logic for bounded branch-width, and [S.-I. Oum, *Approximating rank-width and clique-width quickly*, in Proceedings of the 31st International Workshop on Graph-Theoretic Concepts in Computer Science, Springer-Verlag, Heidelberg, to appear] a cubic time approximation algorithm for graph rank-width and clique-width.

Key words. representable matroid, parametrized algorithm, branch-width, rank-width

AMS subject classifications. 05B35, 68R05

DOI. 10.1137/S0097539702418589

1. Introduction. We assume that the reader is familiar with basic concepts of graph theory; see, for example, [8]. In the past decade, the notion of a *tree-width* of graphs [22] attracted plenty of attention from both graph-theoretical and computational points of view. This attention followed the pioneering work of Robertson and Seymour on the graph minor project [21], and results of various researchers using tree-width in dynamic programming and parametrized complexity; see [3] for a brief introduction. We postpone formal definitions until later sections.

The theory of *parametrized complexity* provides a background for analysis of difficult algorithmic problems and is finer than classical complexity theory. For an overview, we suggest [9]. Briefly, a problem is called “fixed-parameter tractable” if there is an algorithm having running time with the (possible) super-polynomial part separated in terms of some natural “parameter,” which is supposed to be small even for large inputs in practice. Successful practical applications of this concept are known, for example, in computational biology and database theory: Imagine a query of a small size k to a large database of size $n \gg k$; then an $O(2^k \cdot n)$ parametrized algorithm may be better in practice than, say, an $O(n^k)$ algorithm, or even an $O((kn)^c)$ polynomial algorithm.

Generally speaking, we are interested in algorithmic problems that are parametrized by a tree-like structure of the input objects or the tree-width parameter. However, for matroid theorists, it is the (very similar, but less known) parameter called branch-width [22] that has proved to be the more useful tool. This is because, unlike tree-width, branch-width does not refer to vertices, and thus extends directly from graphs to matroids, and hence also to matrices and vector configurations. We refer to [16] for a more detailed discussion on matroid tree-width.

*Received by the editors November 25, 2002; accepted for publication (in revised form) April 15, 2005; published electronically October 7, 2005. This paper is based on the author’s research originally carried out at the Victoria University of Wellington in New Zealand, 2001–2002. The author also acknowledges generous research support given by the NZ Marsden Fund (a grant to Geoff Whittle) in 2000–2002, and partial support of Czech research grant GAČR 201/05/0050.

<http://www.siam.org/journals/sicomp/35-2/41858.html>

†Department of Computer Science, FEI, VŠB—Technical University of Ostrava, 17. listopadu 15, 708 33 Ostrava, Czech Republic (hlineny@member.ams.org, petr.hlineny@vsb.cz).

The tree-width of a graph is hard to compute in general. It has been shown by Bodlaender [4] that graph tree-width is a uniformly fixed-parameter tractable property, and, moreover, an optimal tree-decomposition of a graph can be found in parametrized linear time. That algorithm is used in [5] to find an optimal branch-decomposition of a graph in parametrized linear time. However, these algorithms are so closely tied to graphs that it seems impossible to generalize them to matroids. Recent advances concerning matroid connectivity, on the other hand, lead to a simple and practical polynomial algorithm [12] for deciding whether a given matroid has branch-width at most 3. Unfortunately, there seems to be no generalization of that algorithm to higher values of branch-width.

In this paper we show (Algorithm 4.1, Theorem 4.14) how to construct a near-optimal branch-decomposition of a matroid represented over any finite field in parametrized (with respect to branch-width) cubic time. We also prove (Theorem 5.5) that matroid branch-width and tree-width are fixed-parameter tractable properties for matroids represented over finite fields.

These algorithms and their consequences are formulated in the language of matroid theory since it is natural and convenient, and since it shows the close relations of this research to well-known graph structural and computational concepts. Our work could be, as well, viewed in terms of matrices, point configurations, or linear codes over a finite field \mathbb{F} . The key to the subject is the notion of parse trees for bounded-width \mathbb{F} -represented matroids, defined in section 3 as an analogue of parse trees for graphs of bounded tree-width.

In [14] we prove a result analogous to the so-called “ MS_2 -theorem” of Courcelle [6, 7] for matroids represented by matrices over a finite field \mathbb{F} : If \mathcal{M} is a family of matroids described by a sentence in the monadic second-order logic of matroids, then the “parse trees” of bounded-branch-width \mathbb{F} -represented members of \mathcal{M} are recognizable by a finite tree automaton. So by relating [14] to Algorithm 4.1, we prove that all matroid properties expressible in the monadic second-order logic are uniformly fixed-parameter tractable for \mathbb{F} -represented matroids of bounded branch-width. Another application [13] gives an efficient algorithm for computing the Tutte polynomial of a matroid, the critical index, and the Hamming weight or the weight enumerator of a linear code, when the branch-width is bounded. Moreover, a recent algorithm of Oum [19] uses Algorithm 4.1 to approximate the clique-width of a graph in parametrized cubic time via a new parameter related to matroid branch-width called the rank-width [18].

In order to make the paper accessible to a wide audience of computer scientists, we provide sufficient introductory definitions for relevant concepts of structural matroid theory.

2. Basics of matroids. We refer to Oxley [20] for our matroid terminology. A *matroid* is a pair $M = (E, \mathcal{B})$, where $E = E(M)$ is the ground set of M (elements of M), and $\mathcal{B} \subseteq 2^E$ is a nonempty collection of *bases* of M , no two of which are in an inclusion. Moreover, matroid bases satisfy the “exchange axiom”; if $B_1, B_2 \in \mathcal{B}$ and $x \in B_1 - B_2$, then there is $y \in B_2 - B_1$ such that $(B_1 - \{x\}) \cup \{y\} \in \mathcal{B}$. Subsets of bases are called *independent sets*, and the remaining sets are *dependent*. Minimal dependent sets are called *circuits*. The *rank function* $r_M(X)$ in M is the maximal cardinality of an independent subset of a set $X \subseteq E(M)$.

If G is a graph, then its *cycle matroid* on the ground set $E(G)$ is denoted by $M(G)$. The independent sets of $M(G)$ are the forests of G , and the circuits of $M(G)$ are the cycles of G . In fact, much of matroid terminology is inherited from graphs. Another typical example of a matroid is a finite set of vectors with usual linear dependency.

Both examples are illustrated in Figure 1.

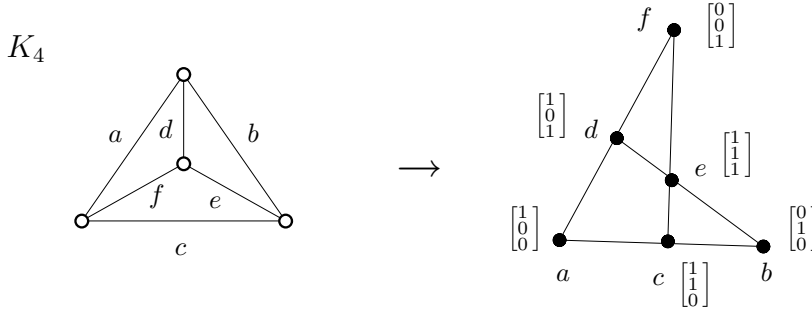


FIG. 1. An example of a vector representation of the cycle matroid $M(K_4)$. The matroid elements are depicted by dots, and their (linear) dependency is shown using lines.

The dual matroid M^* of M is defined on the same ground set E , and the bases of M^* are the set-complements of the bases of M . An element e of M is called a loop (a coloop) if $\{e\}$ is dependent in M (in M^*). The matroid $M \setminus e$ obtained by deleting a nonloop element e is defined as $(E - \{e\}, \mathcal{B}^-)$, where $\mathcal{B}^- = \{B : B \in \mathcal{B}, e \notin B\}$. The matroid M/e obtained by contracting a nonloop element e is defined using duality $M/e = (M^* \setminus e)^*$. (This corresponds to contracting an edge in a graph.) A minor of a matroid is obtained by a sequence of deletions and contractions of elements.

The connectivity function λ_M of a matroid M is defined for all $A \subseteq E$ by

$$\lambda_M(A) = r_M(A) + r_M(E - A) - r(M) + 1.$$

Here $r(M) = r_M(E)$. Notice that $\lambda_M(A) = \lambda_M(E - A)$. A subset $A \subseteq E$ is k -separating if $\lambda_M(A) \leq k$. When equality holds here, A is said to be exactly k -separating. An arbitrary partition $(A, E - A)$ of M is called a separation in M . A partition $(A, E - A)$ is called a k -separation if A is k -separating and both $|A|, |E - A| \geq k$. Geometrically, the affine closures of the two sides of an exact k -separation intersect in a subspace of rank $k - 1$ (such as in a line if $k = 3$).

2.1. Branch-decomposition. A subcubic tree is a tree in which all nodes have degree at most 3. (We do not use the word ternary because such trees are actually subbinary in the sense of the next section.) Let $\ell(T)$ denote the set of leaves of a tree T . The next definition of branch-width for matroids directly extends branch-width of graphs.

Let M be a matroid on the ground set $E = E(M)$. A branch-decomposition of M is a pair (T, τ) , where T is a subcubic tree, and τ is an injection of E into $\ell(T)$, called labeling. Let e be an edge of T , and let T_1, T_2 be the connected components of $T - e$. We say the e displays the partition (A, B) of E , where $A = \tau^{-1}(\ell(T_1))$, $B = \tau^{-1}(\ell(T_2))$. The width of an edge e in T is $\omega_T(e) = \lambda_M(A) = \lambda_M(B)$. The width of the branch-decomposition (T, τ) is the maximum of the widths of all edges of T , and the branch-width of M is the minimal width over all branch-decompositions of M . If T has no edge, then we take its width as 0.

Examples of branch-decompositions are presented in Figure 2. We remark that the branch-width of a graph is defined analogously, using the connectivity function λ_G , where $\lambda_G(F)$ is the number of vertices incident with both F and $E(G) - F$. Clearly, the branch-width of a graph G is never smaller than the branch-width of its

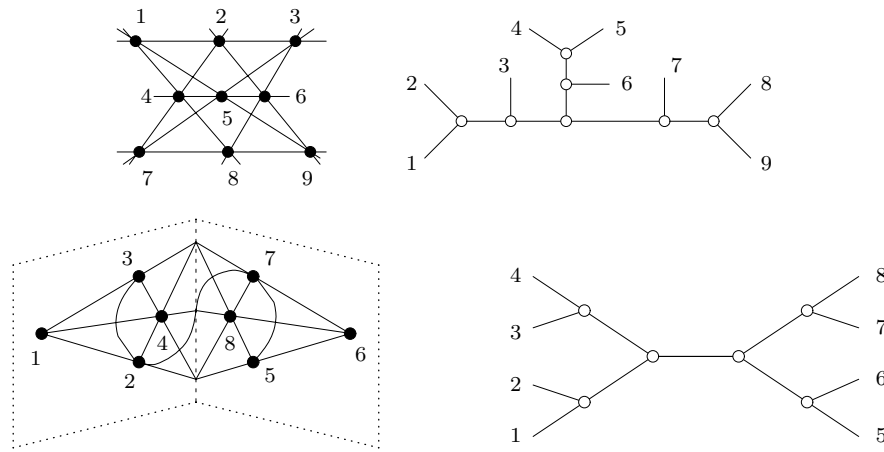


FIG. 2. Two examples of width-3 branch decompositions of the Pappus matroid (top left, in rank 3) and of the binary affine cube (bottom left, in rank 4). The lines in the matroids show dependencies among elements.

cycle matroid $M(G)$. It is still an open conjecture whether these numbers are actually equal. On the other hand, branch-width is within a constant factor of tree-width on graphs [22] and also on matroids [16].

2.2. Represented matroids. We now turn our attention to matroids represented over a fixed finite field \mathbb{F} . This is a crucial part of our introductory definitions. A *representation* of a matroid M is a matrix \mathbf{A} whose column vectors correspond to the elements of M , and maximal linearly independent subsets of columns form the bases of M . We denote by $M(\mathbf{A})$ the matroid represented by a matrix \mathbf{A} .

We denote by $PG(n, \mathbb{F})$ the *projective geometry (space)* obtained from the vector space \mathbb{F}^{n+1} . See [20, sections 6.1, 6.3] for an overview of projective spaces and of (in)equivalence of matroid representations. For a set $X \subseteq PG(n, \mathbb{F})$, we denote by $\langle X \rangle$ the span (affine closure) of X in the space. The (projective) rank $r(X)$ of X is the maximal cardinality of a linearly independent subset of X . A projective transformation is a mapping between two projective spaces over \mathbb{F} that is induced by a linear transformation between the underlying vector spaces. Clearly, the matroid $M(\mathbf{A})$ represented by a matrix \mathbf{A} is unchanged when column vectors are scaled by nonzero elements of \mathbb{F} . Hence we may view a loopless matroid representation $M(\mathbf{A})$ as a multiset of points in the finite projective space $PG(n, \mathbb{F})$, where n is the rank of $M(\mathbf{A})$.

DEFINITION 2.1. We call a finite multiset of points in a projective space over \mathbb{F} a point configuration, and we represent a loop in a point configuration by the empty subspace \emptyset . Two point configurations P_1, P_2 in projective spaces over \mathbb{F} are equivalent if there is a nonsingular projective transformation between the projective spaces that maps P_1 onto P_2 bijectively. (Loops are mapped only to loops.) We define an \mathbb{F} -represented matroid to be such an equivalence class of point configurations over \mathbb{F} .

One may think that we do not have to include the word “bijectively” in the previous definition since nonsingular projective transformations are always injective on the points, but, in fact, we have to do this to handle multiple elements in multisets. Two labeled point configurations over \mathbb{F} are equivalent in our sense if and only if, in the language of [20, Chapter 6], the matrix representations are equivalent without use

of \mathbb{F} -automorphisms, otherwise called *strongly equivalent* in matroid theory. However, notice that our represented matroids are unlabeled in general.

Standard matroidal terms are inherited from matroids to represented matroids. Obviously, all point configurations in one equivalence class belong to the same isomorphism class of matroids, but the converse is not true in general since matroids often have inequivalent representations. When we want to deal with an \mathbb{F} -represented matroid, we actually pick an arbitrary point configuration from the equivalence class.

3. Parse trees for matroids. In this section we introduce our basic formal tool—the parse trees for represented matroids of bounded branch-width. Loosely speaking, a parse tree shows how to “build up” a matroid along the tree using only a fixed amount of information at each tree node, and so it forms a suitable background for dynamic programming.

We are inspired by analogous boundaried graphs and parse trees known for handling graphs of bounded tree-width (see for example [1] or [9, section 6.4]): A boundaried graph is a graph with a distinguished subset of labeled vertices. (The purpose of distinguishing the boundary is that only the boundary vertices can be “accessed from outside”). Then, simply speaking, a graph has tree-width at most $t - 1$ if and only if it can be composed from small pieces by gluing them onto boundaries of size at most t . We similarly define boundaried represented matroids, in which the boundary is a distinguished subspace of the representation, and composition operators that are used to glue representations together. (The role of composition operators, however, differs between tree-width and branch-width.)

A *rooted ordered subbinary tree* is such that each of its nodes has at most two sons that are ordered as “left” and “right.” (If there is one son, then it may be either left or right.) A *rooted subtree* T_0 of a rooted tree T is a subgraph of T such that T_0 is the connected component of $T - e$ not containing the root for some $e \in E(T)$. Let Σ be a finite alphabet. We denote by Σ^{**} the class of rooted ordered subbinary trees with nodes labeled by symbols from Σ .

3.1. Boundaried matroids. All matroids throughout this section are \mathbb{F} -represented for some fixed finite field \mathbb{F} . Hence, for simplicity, if we say “(represented) matroid,” then we mean an \mathbb{F} -represented matroid. If we speak about a projective space, we mean a projective geometry over the field \mathbb{F} . Let $[s, t]$ denote the set $\{s, s + 1, \dots, t\}$.

The following definition presents a possible way of formalizing the notion of a “matroid with a boundary.” (Since matroids have no vertices, unlike graphs, we have to introduce some special elements that define the matroid boundary.)

DEFINITION 3.1. A pair $\bar{N} = (N, \delta)$ is a t -boundaried (represented) matroid if the following holds: $t \geq 0$ is an integer, N is a represented matroid, and $\delta : [1, t] \rightarrow E(N)$ is an injective mapping such that $\delta([1, t])$ is an independent set in N .

We call $J(\bar{N}) = E(N) - \delta([1, t])$ the *internal elements* of \bar{N} , elements of $\delta([1, t])$ the *boundary points* of \bar{N} , and t the *boundary rank* of \bar{N} . The represented matroid $N \setminus \delta([1, t])$, which is the restriction of N to $J(\bar{N})$, is called the *internal matroid* of \bar{N} . We denote by $\partial(\bar{N})$ the boundary subspace spanned by $\delta([1, t])$. In particular, the boundary points are not loops. The basic operation we use is the *boundary sum* \oplus of the next definition, illustrated in Figure 3.

DEFINITION 3.2. Let $\bar{N}_1 = (N_1, \delta_1)$, $\bar{N}_2 = (N_2, \delta_2)$ be two t -boundaried represented matroids. We denote by $\bar{N}_1 \oplus \bar{N}_2 = N$ the represented matroid defined as follows: Let Ψ_1, Ψ_2 be projective spaces such that the intersection $\Psi_1 \cap \Psi_2$ has rank exactly t . Suppose that, for $i = 1, 2$, $P_i \subset \Psi_i$ is a point configuration representing N_i

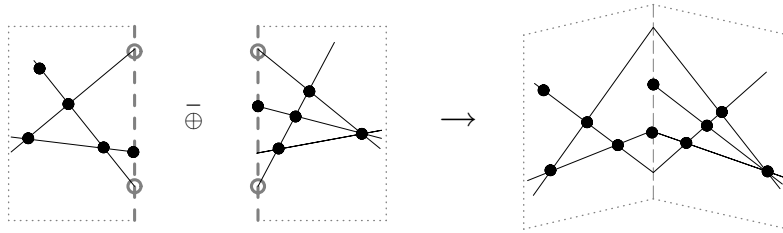


FIG. 3. An example of a boundary sum of two 2-boundaried matroids. The internal matroid elements are drawn as solid dots; the boundary points and the boundary subspace of rank 2 are drawn in gray. Solid lines show matroid dependencies. The resulting sum is a matroid represented on two intersecting planes in rank 4 (three-dimensional picture on the right).

such that $P_1 \cap P_2 = \delta_1([1, t]) = \delta_2([1, t])$, and $\delta_2(j) = \delta_1(j)$ for $j \in [1, t]$. Then N is the matroid represented by $(P_1 \cup P_2) - \delta_1([1, t])$.

Informally, the boundary sum $\bar{N}_1 \oplus \bar{N}_2 = N$ on the ground set $E(N) = J(\bar{N}_1) \dot{\cup} J(\bar{N}_2)$ is obtained by gluing the representations of N_1 and N_2 onto a common subspace (the boundary) of rank t so that the boundary points of both are identified in order and then deleted. Keep in mind that a point configuration is a multiset. It is a matter of elementary linear algebra to verify that the boundary sum is well defined with respect to equivalence of point configurations.

We write “ $\leq t$ -boundaried” to mean t' -boundaried for some $0 \leq t' \leq t$. We now define a composition operator (over the field \mathbb{F}) which will be used to generate large boundaried matroids from smaller pieces (Figure 4).

DEFINITION 3.3. A $\leq t$ -boundaried composition operator is defined as a quadruple $\odot = (R, \gamma_1, \gamma_2, \gamma_3)$, where R is a represented matroid, $\gamma_i : [1, t_i] \rightarrow E(R)$ is an injective mapping for $i = 1, 2, 3$ and some fixed $0 \leq t_i \leq t$, each $\gamma_i([1, t_i])$ is an independent set in R , and $(\gamma_i([1, t_i]) : i = 1, 2, 3)$ is a partition of $E(R)$.

The $\leq t$ -boundaried composition operator \odot is a binary operator applied to a t_1 -boundaried represented matroid $\bar{N}_1 = (N_1, \delta_1)$ and to a t_2 -boundaried represented matroid $\bar{N}_2 = (N_2, \delta_2)$. The result of the composition is a t_3 -boundaried represented matroid $\bar{N} = (N, \gamma_3)$, written as $\bar{N} = \bar{N}_1 \odot \bar{N}_2$, where a matroid N is defined using boundaried sums: $N' = \bar{N}_1 \oplus (R, \gamma_1)$, $N = (N', \gamma_2) \oplus \bar{N}_2$.

Speaking informally, a boundaried composition operator is a bounded-rank configuration with three boundaries distinguished by $\gamma_1, \gamma_2, \gamma_3$ and with no other internal points. For reference we denote $t_i(\odot) = t_i$, $R(\odot) = R$, and $\gamma_i(\odot) = \gamma_i$. The meaning of a composition $\bar{N} = \bar{N}_1 \odot \bar{N}_2$ is that, for $i = 1, 2$, we glue the represented matroid N_i onto R , matching $\delta_i([1, t_i])$ with $\gamma_i([1, t_i])$ in order. The result is a t_3 -boundaried matroid \bar{N} with boundary $\gamma_3([1, t_3])$. One may abbreviate the composition as $\bar{N} = ((\bar{N}_1 \oplus (R, \gamma_1), \gamma_2) \oplus \bar{N}_2, \gamma_3)$.

3.2. Parse trees. The main purpose of introducing parse trees is in that they allow us to formally define how to construct a represented matroid of branch-width at most $t + 1$ using $\leq t$ -boundaried composition operators.

Let $\bar{\Omega}_t$ denote the empty t -boundaried matroid (Ω, δ_0) , where $t \geq 0$ and $\delta_0([1, t]) = E(\Omega)$ (t will often be implicit in the context). If $\bar{N} = (N, \delta)$ is an arbitrary t -boundaried matroid, then $\bar{N} \oplus \bar{\Omega}_t$ is actually the internal matroid of \bar{N} . Let $\bar{\Upsilon}$ denote the single-element 1-boundaried matroid (Υ, δ_1) , where $E(\Upsilon) = \{x, x'\}$ are two parallel elements and $\delta_1(1) = x'$. Let $\bar{\Upsilon}_0$ denote the loop 0-boundaried matroid (Υ_0, δ_0) ,

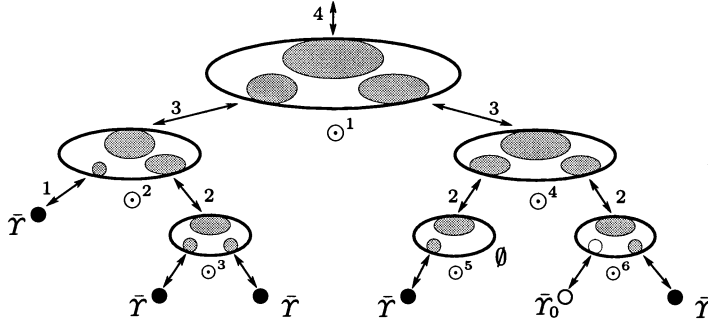


FIG. 4. An example of a bounded parse tree. The ovals represent composition operators, with shaded parts for the boundaries and edge-numbers for the boundary ranks (e.g., $\odot^4 = (R^4, \gamma_1^4, \gamma_2^4, \gamma_3^4)$, where $\gamma_1^4, \gamma_2^4 : [1, 2] \rightarrow E(R^4)$, $\gamma_3^4 : [1, 3] \rightarrow E(R^4)$).

where $E(\Upsilon_0) = \{z\}$ is a loop and δ_0 is an empty mapping. Let $\mathcal{R}_t^{\mathbb{F}}$ denote the finite set of all $\leq t$ -bounded composition operators over the field \mathbb{F} . We set $\Pi_t = \mathcal{R}_t^{\mathbb{F}} \cup \{\tilde{\Upsilon}, \tilde{\Upsilon}_0\}$.

Let $T \in \Pi_t^{**}$ be a rooted ordered subbinary tree with nodes labeled by the alphabet Π_t . Considering a node v of T , we set $\varrho(v) = 1$ if v is labeled by $\tilde{\Upsilon}$, $\varrho(v) = 0$ if v is labeled by $\tilde{\Upsilon}_0$, and $\varrho(v) = t_3(\odot)$ if v is labeled by \odot . We call T a $\leq t$ -bounded parse tree if the following are true:

- Only leaves of T are labeled by $\tilde{\Upsilon}$ or $\tilde{\Upsilon}_0$.
- If a node v of T labeled by a composition operator \odot has no left (no right) son, then $t_1(\odot) = 0$ ($t_2(\odot) = 0$).
- If a node v of T labeled by \odot has left son u_1 (right son u_2), then $t_1(\odot) = \varrho(u_1)$ ($t_2(\odot) = \varrho(u_2)$).

Informally, the boundary ranks of composition operators and/or single-element terminals must “agree” across each edge. Notice that $\tilde{\Upsilon}$ or $\tilde{\Upsilon}_0$ are the only labels from Π_t that “create” elements of the resulting represented matroid $P(T)$ in the next definition. See an illustrative example in Figure 4.

DEFINITION 3.4. Let T be a $\leq t$ -bounded parse tree. The $\leq t$ -bounded represented matroid $\bar{P}(T)$ parsed by T is recursively defined as follows:

- If T is an empty tree, then $\bar{P}(T) = \bar{\Omega}_0$.
- If T has one node labeled by $\tilde{\Upsilon}$ (by $\tilde{\Upsilon}_0$), then $\bar{P}(T) = \tilde{\Upsilon}$ ($= \tilde{\Upsilon}_0$).
- If the root r of T is labeled \odot^r , and r has a left rooted subtree T_1 and a right rooted subtree T_2 (possibly empty), then $\bar{P}(T) = \bar{P}(T_1) \odot^r \bar{P}(T_2)$.

The composition is well defined according to the parse-tree description in the previous paragraph. The represented matroid parsed by T is $P(T) = \bar{P}(T) \oplus \bar{\Omega}$.

We say that a t -bounded represented matroid \bar{M} is spanning if the boundary subspace $\partial(\bar{M})$ is contained in the span $\langle J(\bar{M}) \rangle$ of the internal points of \bar{M} . We say that a $\leq t$ -bounded parse tree T is spanning if, for each nonempty rooted subtree T_1 of T , the bounded matroid $\bar{P}(T_1)$ is spanning and nonempty. The following natural result about parse trees of matroids is proved in [14].

THEOREM 3.5 (Hliněný [14]). An \mathbb{F} -represented matroid M has branch-width at most $t + 1$ if and only if M is parsed by some spanning $\leq t$ -bounded parse tree.

It is easy to turn a bounded parse tree into a branch-decomposition. Conversely, the proof of Theorem 3.5 shows how to construct a bounded parse tree from a given branch-decomposition. It is, however, more efficient to construct a bounded parse tree directly from scratch and that is what we are going to do in this paper.

4. Constructing a parse tree. We start with an overview of our algorithm for construction of a matroid parse tree. We assume that a matroid M is given by a matrix $\mathbf{A} \in \mathbb{F}^{r \times n}$ of rank r over a finite field \mathbb{F} . The size of M is expressed in terms of the number n of elements, i.e., the number of columns of \mathbf{A} . Clearly, $r \leq n$. Additionally, there are two *parameters*—a finite field \mathbb{F} and an integer $t > 0$. These parameters form a separate part of the input in our algorithm, but they are considered constants for the purpose of complexity analysis of the algorithm. A more formal description in the scope of parametrized complexity can be found in section 5.2.

We call a labeled tree T a *partial $\leq t$ -boundaried parse-tree* if T satisfies all properties of the parse-tree definition (section 3.2) except that some leaves of T may be labeled by arbitrary t' -boundaried represented matroids for $t' \leq t$. The matroid $\bar{P}(T)$ parsed by T is defined analogously to ordinary parse trees.

Notice that a 0-boundaried represented matroid is essentially an ordinary represented matroid, and that the boundary sum of 0-boundaried represented matroids is the ordinary direct sum of matroids. Let $M \upharpoonright X$ denote the restriction of a represented matroid M to a subset of elements $X \subseteq E(M)$. Let \mathbf{I}_r denote the $r \times r$ identity submatrix.

ALGORITHM 4.1. *Computing a spanning boundaried parse tree of a matroid:*

Parameters: A given finite field \mathbb{F} , and an integer $t \geq 1$.

Input: A matrix $\mathbf{A} = [\mathbf{I}_r \mid \mathbf{A}'] \in \mathbb{F}^{r \times n}$ given in the standard form over \mathbb{F} , such that the matroid $M(\mathbf{A})$ represented by \mathbf{A} has branch-width at most $t + 1$.

1. Let $M = M(\mathbf{A})$, and let $E(M) = B \cup F$, where a basis B marks the columns of \mathbf{I}_r and F the columns of \mathbf{A}' . We initially set $X = B$, and set T to be an arbitrary 0-boundaried parse tree for the independent matroid $M \upharpoonright B$.
2. For an arbitrary element $f \in F - X$, we set $X = X \cup \{f\}$, and we add a new single-element leaf representing f in $M \upharpoonright X$ to the parse tree T .
3. If the width of the parse tree T exceeds, say, $10t$, or if $X \supseteq F$, then we compute a new spanning $\leq 3t$ -boundaried parse tree T' for $M \upharpoonright X$ using T :
 - (a) We start with T' equal to the trivial 0-boundaried partial parse tree having one node labeled by $M \upharpoonright X$.
 - (b) Let ℓ be a leaf of T' labeled by a boundaried matroid \bar{N} with more than one internal element. If the boundary rank of \bar{N} is less than $3t$, then we choose an arbitrary internal element e in \bar{N} . We add to T' a new leaf ℓ_1 representing a single-element e , another new leaf ℓ_2 labeled by $\bar{N} \setminus e$, and we relabel ℓ with the corresponding composition operator.
 - (c) If the boundary rank of \bar{N} equals $3t$, then there are two $\leq 3t$ -boundaried matroids \bar{N}_1, \bar{N}_2 such that $\bar{N} = \bar{N}_1 \odot \bar{N}_2$ for some $\leq 3t$ -boundaried composition operator \odot . Using the decomposition T , we can find \bar{N}_1, \bar{N}_2 and \odot efficiently. Then we add two new leaves ℓ_1, ℓ_2 to T' labeled by \bar{N}_1, \bar{N}_2 , respectively, and we relabel ℓ with \odot .
 - (d) We repeat steps 3(b)–(c) until T' is an ordinary parse tree.

Finally, we set $T = T'$.

4. We repeat steps 2–3 until $X \supseteq F$.

Output: A spanning $\leq 3t$ -boundaried parse tree $T \in \Pi_{3t}^{**}$ such that the represented matroid $P(T)$ parsed by T is equal to $M(\mathbf{A})$.

Remark 4.2. If the above promise that the matroid $M(\mathbf{A})$ has branch-width at most $t + 1$ is false, then step 3(c) may fail to find \bar{N}_1 and \bar{N}_2 . In such a case, Algorithm 4.1 ends up with an error. The algorithm may be formulated so that it always finds a spanning $\leq 3t$ -boundaried parse tree T for $M(\mathbf{A})$ if branch-width of

$M(\mathbf{A})$ is at most $t + 1$, but the output is (possibly) empty if branch-width of $M(\mathbf{A})$ exceeds $t + 1$.

Remark 4.3. Another point concerns the given finite field \mathbb{F} : A finite field is uniquely determined by the number of its elements $q = |\mathbb{F}|$. Moreover, it is easy to construct addition and multiplication tables for \mathbb{F} algorithmically from given q .

We claim that Algorithm 4.1 finds the $\leq 3t$ -boundaried parse tree T for $M(\mathbf{A})$ in time $O(n^3)$, where n is the number of columns of \mathbf{A} . If the matrix \mathbf{A} is not in the standard form, we can easily get the standard form in time $O(n^3)$. Step (4.1) can be implemented in time $O(nr) \leq O(n^2)$. Each iteration of the cycle in step (4.1) can be implemented in time $O(n)$, and there are at most $n - 1$ iterations. Altogether, the main cycle in the algorithm is repeated $n - r$ times.

We also note that the running time $O(n^3)$ of our algorithm refers to the number n of columns of \mathbf{A} , not to the real size of the input. The real input size of the matrix \mathbf{A} is $O(n \cdot r)$, which is typically of order up to n^2 . We do not attempt to determine how the running time of Algorithm 4.1 depends on the parameters \mathbb{F} and t (which is at least an exponential function). However, we do care that the algorithm is recursive in \mathbb{F} and t —there is one algorithm, *not* a sequence of algorithms, running for all values of the parameters (cf. section 5.2).

4.1. Step 4.1: Adding an element to a parse tree. Step 4.1 in Algorithm 4.1 is easy to implement. We are going to describe the implementation of step 4.1 of Algorithm 4.1. This is a surprisingly nontrivial task despite our adding the element f to the parse tree T arbitrarily. The main complication comes from the necessity of recomputing all boundary subspaces of the separations in T —a straightforward implementation of which would require us to solve systems of linear equations. All of our succeeding algorithms are parametrized by a finite field \mathbb{F} and integers t and $t' \leq 10t$, where \mathbb{F} , t are as in Algorithm 4.1.

Let $M = M(\mathbf{D})$ be a matroid represented by a matrix $\mathbf{D} \in \mathbb{F}^{r \times n}$, and consider a separation (E_1, E_2) in M (i.e., a partition of $E(M)$). The projective subspace $\langle E_1 \rangle \cap \langle E_2 \rangle$ spanned by both sides of the separation is called the *guts of the separation* (E_1, E_2) . The rank of the guts equals $\lambda_M(E_1) - 1$ by modularity. Naturally, one may compute a spanning set of *generator vectors* (with respect to \mathbf{D}) for the guts of (E_1, E_2) from the vectors in \mathbf{D} . If the rank of the guts is less than t' , which is a constant, then the combined size of its independent generator vectors is $O(r)$.

ALGORITHM 4.4 (adding a vector to separation guts in a represented matroid).

Input: A matrix $\mathbf{D} = [\mathbf{I}_r \mid \mathbf{D}'] \in \mathbb{F}^{r \times n}$ representing the matroid $M = M(\mathbf{D})$; a separation (F_1, F_2) of the matroid $M \setminus f$, where f is the element represented by the last column of \mathbf{D} and where $\lambda_{M \setminus f}(F_1) \leq t' + 1$; and independent generator vectors for the guts of the separation (F_1, F_2) with respect to $\mathbf{D} \setminus f$.

Output: Independent generator vectors for the guts of the separation $(F_1 \cup \{f\}, F_2)$ of M with respect to \mathbf{D} , computed in time $O(r)$.

Proof. Notice that we cannot even read the whole matrix \mathbf{D} in time $O(r)$. This is, however, not a big problem since we are going to use only the unit vectors of \mathbf{I}_r and the generator vectors of the guts in the algorithm. Let $\Psi = \langle F_1 \rangle \cap \langle F_2 \rangle$ be the given guts, and let $\Psi' = \langle F_1 \cup \{f\} \rangle \cap \langle F_2 \rangle$ be the guts we have to generate. Let \mathbf{f} denote the last column of \mathbf{D} representing the element f , and let \mathbf{f}_1 denote the vector obtained from \mathbf{f} by setting to 0 the coordinates corresponding to the unit vectors of \mathbf{I}_r appearing in F_1 . Then \mathbf{f}_1 belongs to the span $\langle F_2 \rangle$ by definition.

We may easily compute \mathbf{f}_1 in time $O(r)$. Moreover, we may decide whether $\mathbf{f}_1 \in \Psi$ in time $O(r)$ since there is a bounded number of generator vectors for Ψ . If

$\mathbf{f}_1 \in \Psi$, then \mathbf{f} belongs to the span $\langle F_1 \rangle$ and so $\Psi' = \Psi$. Otherwise, since \mathbf{f}_1 belongs to the span $\langle F_1 \cup \{f\} \rangle$, we get $\Psi' = \langle \Psi \cup \mathbf{f}_1 \rangle$. \square

Let $M = M(\mathbf{D})$ be a matroid represented by a matrix $\mathbf{D} \in \mathbb{F}^{r \times n}$, and let T be a spanning $\leq t'$ -boundaried parse tree for M . A *coordinatization* (with respect to \mathbf{D}) of the parse tree T is the assignment of the appropriate vectors to the boundary points of the matroids $\bar{P}(T_1)$ for all rooted subtrees T_1 of T , as computed from the vectors of \mathbf{D} . Similarly as above, the combined size of the vectors for each boundary subspace is $O(r)$.

ALGORITHM 4.5 (computing the coordinatization of a spanning $\leq t'$ -boundaried parse tree T for the represented matroid $M(\mathbf{D})$ over \mathbb{F}).

Input: A matrix $\mathbf{D} \in \mathbb{F}^{r \times n}$, and a spanning $\leq t'$ -boundaried parse tree T for the represented matroid $M(\mathbf{D})$.

Output: The coordinatization of T with respect to \mathbf{D} , computed in time $O(nr)$.

Proof. If T_0 is a rooted subtree of T with one node, then the coordinatization of T_0 is trivial. So assume that the root r_0 of T_0 has the left and right rooted subtrees T_1 and T_2 , and that r_0 is labeled by a composition operator \odot . (Hence $\bar{P}(T_0) = \bar{P}(T_1) \odot \bar{P}(T_2)$. One of the subtrees may be empty.) Since T is a spanning parse tree, the boundary of $\bar{P}(T_0)$ is spanned by the boundaries of $\bar{P}(T_1)$ and $\bar{P}(T_2)$. Thus the vectors assigned to the boundary points of $\bar{P}(T_0)$ with respect to \mathbf{D} are linear combinations of the vectors of the boundary points of $\bar{P}(T_1)$ and $\bar{P}(T_2)$. There are finitely many $\leq t'$ -boundaried composition operators for a fixed t' , and so the scalars of these linear combinations can be precomputed for the parameter t' and each \odot . Then, at each node of T , we have to compute a bounded number of linear combinations from a bounded number of vectors of length r , which can be accomplished in time $O(r)$. There are $O(n)$ nodes in T . \square

Consider three sequences $U_1, U_2, U_3 \subset \mathbb{F}^s$ of vectors, where each $U_i, i = 1, 2, 3$, is formed by independent vectors. Then these sequences represent the composition operator $(U, \gamma_1, \gamma_2, \gamma_3)$, where $U = U_1 \cup U_2 \cup U_3$ is the point configuration, and γ_i maps the elements of the sequence in order $U_i = \{\gamma_i(1), \gamma_i(2), \dots, \gamma_i(t_i)\}$ for $i = 1, 2, 3$.

ALGORITHM 4.6 (computing the composition operator from given vectors).

Input: Three independent sequences $U_1, U_2, U_3 \subset \mathbb{F}^s$ of vectors where each sequence has at most t' members.

Output: The $\leq t'$ -boundaried composition operator represented by the vectors of U_1, U_2, U_3 , computed in time $O(s)$.

Proof. There is a bounded number of $\leq t'$ -boundaried composition operators for each t' , and since each one is nothing else than a labeled point configuration, it may be identified by a bounded number of homogeneous linear vector equations and inequations. (The equations are invariant under nonsingular linear transformations.) Hence we find the composition operator represented by (U_1, U_2, U_3) in parametrized time $O(s)$ even by brute force. \square

LEMMA 4.7. Step 4.1 in Algorithm 4.1 is implemented in time $O(n \cdot r)$ for fixed parameters t' and \mathbb{F} .

Proof. Let $N = M \upharpoonright X$ in step 4.1 of Algorithm 4.1 (before adding f to X). We first compute the coordinatization of the parse tree T for N using Algorithm 4.5. Let T_1 be a rooted subtree of T , and let $F_1 = J(\bar{P}(T_1))$, $F_2 = E(N) - F_1$. Then, since the parse tree T is spanning, the boundary subspace of $\bar{P}(T_1)$ make up the guts of the separation (F_1, F_2) . We denote by T' the rooted subbinary tree obtained from T by arbitrarily adding a new leaf ℓ representing the element f . To turn T' into a parse tree for $N' = M \upharpoonright X \cup \{f\}$, we have to recompute all composition operators in T' .

Let T'_1 be the rooted subtree of T' corresponding to T_1 above, and let $F'_1 = J(\bar{P}(T'_1))$, $F'_2 = E(N') - F'_1$. Then $F'_1 = F_1 \cup \{f\}$ and $F'_2 = F_2$ for $\ell \in V(T'_1)$, or $F'_2 = F_2 \cup \{f\}$ and $F'_1 = F_1$ otherwise. We denote by $\mathbf{D} = [\mathbf{I}_r \mid \mathbf{D}']$ the submatrix of the matrix \mathbf{A} from Algorithm 4.1 representing the matroid N' . We apply Algorithm 4.4 to \mathbf{D} , the separation (F_1, F_2) and f , and to the generator vectors of the boundary points of $\bar{P}(T_1)$ computed above, obtaining generator vectors of the boundary of $\bar{P}(T'_1)$. We repeat the same procedure for all $O(n)$ rooted subtrees of T' . Finally, we use Algorithm 4.6 to determine the new composition operators to label all internal nodes of T' . \square

4.2. Step 4.1: Getting a better parse tree. The heart of the implementation of step 4.1 in Algorithm 4.1 is the next claim. (We remark that the proof of this claim implicitly uses a so-called “tangle” [22]—a notion dual to a branch-decomposition.)

LEMMA 4.8. *Let $t \geq 1$ and \bar{N} be a spanning $3t$ -boundaried represented matroid such that the branch-width of the internal matroid $\bar{N} \oplus \bar{\Omega}$ is at most $t + 1$. Then there are two $\leq 3t$ -boundaried matroids \bar{N}_1, \bar{N}_2 such that $\bar{N} = \bar{N}_1 \odot \bar{N}_2$ for some $\leq 3t$ -boundaried composition operator \odot .*

Proof. Recall that the internal matroid $N' = \bar{N} \oplus \bar{\Omega}$ is the restriction of \bar{N} to the internal elements $J(\bar{N}) = E(N')$. We define a function $g : 2^{E(N')} \rightarrow \mathbb{Z}$, for a subset $F \subseteq E(N')$, as $g(F) = r(\partial(\bar{N}) \cap \langle F \rangle)$, i.e., as the projective rank of the intersection of the span of F with the boundary of \bar{N} . Clearly, $g(E(N')) = 3t$ since \bar{N} is spanning.

Let (U, τ) be a width- $(t + 1)$ branch-decomposition of N' . For an edge $e \in E(U)$, we define $F_i(e) = \tau^{-1}(V(U_i))$ for $i = 1, 2$, where U_1 and U_2 are the connected components of $U - e$. Note that $g(F_1(e)) + g(F_2(e)) \geq g(E(N')) = 3t$.

CLAIM. *There is an edge $e \in E(U)$ such that both $g(F_1(e)), g(F_2(e)) \geq t$.*

For a contradiction, we assume that no such edge e exists in U . Let (arbitrary) $e = w_1 w_2$, where $w_i \in U_i$, $i = 1, 2$, as above, and $a = \min \{g(F_1(e)), g(F_2(e))\}$. If $a < t$, then, say, $a = g(F_1(e)) < g(F_2(e))$. In such a case we direct the edge e from w_1 to w_2 . Since U is a cubic tree, there is a node w_0 of U such that all three edges incident with w_0 are directed toward it. Then, denoting by U'_1, U'_2, U'_3 the connected components of $U - w_0$ and denoting by $F'_i = \tau^{-1}(V(U'_i))$, we get $g(F'_1) + g(F'_2) + g(F'_3) < t + t + t$, which is a contradiction to $g(E(N')) = 3t$.

We consider further the edge e from the claim. The rank of $\langle F_1(e) \rangle \cap \langle F_2(e) \rangle$ is at most t since (U, τ) is a width- $(t + 1)$ branch-decomposition. Together for $i = 1, 2$ we get, by modularity of the rank in projective spaces,

$$\begin{aligned} & r[\langle F_i(e) \rangle \cap \langle F_{3-i}(e) \cup \partial(\bar{N}) \rangle] \\ &= r[\langle F_i(e) \rangle \cap \langle F_{3-i}(e) \rangle] + r[\langle F_i(e) \rangle \cap \partial(\bar{N})] - r[\langle F_i(e) \rangle \cap \langle F_{3-i}(e) \rangle \cap \partial(\bar{N})] \\ &\leq t + 2t - 0 = 3t. \end{aligned}$$

Hence the partition $(F_1(e), F_2(e))$ decomposes the internal elements into two $\leq 3t$ -boundaried matroids \bar{N}_1, \bar{N}_2 , which are glued together by a suitable $\leq 3t$ -boundaried composition operator \odot . \square

The task now is to find the partition (F_1, F_2) of $J(\bar{N})$ inducing the boundaried matroids \bar{N}_1, \bar{N}_2 and to find the composition operator \odot from Lemma 4.8 efficiently. We use the $\leq t'$ -boundaried parse tree T previously constructed in Algorithm 4.1. Unlike in the implementation of step 4.1, we do not work with the vectors representing $M(\mathbf{A})$ —instead, we obtain all necessary information from the parse tree T . This results in a more complicated but faster implementation. Again, all algorithms in this section are parametrized by a finite field \mathbb{F} and integers t and $t' \leq 10t$. We first

present the following two simple algorithms.

ALGORITHM 4.9 (computing the connectivity function of a separation).

Input: A $\leq t'$ -boundaried parse tree T parsing the matroid $N = P(T)$, and a partition (F_1, F_2) of $E(N)$.

Output: The connectivity value $\lambda_N(F_1) - 1$ of the separation (F_1, F_2) , computed in time $O(|V(T)|)$.

Proof. In other words, we are computing the projective rank of the guts of separation (F_1, F_2) in N . This is a straightforward application of dynamic programming over T . Let x be a node of T , and let T_x be the rooted subtree of T with the root x . Denote by $\bar{M}_x = \bar{P}(T_x)$ and by T'_x, T''_x the left and right rooted subtrees of x in T .

For \bar{M}_x as above, and for $E_i = F_i \cap J(\bar{M}_x)$, we call *boundary data* the triple (Σ_1, Σ_2, g) , where $\Sigma_i = \partial(\bar{M}_x) \cap \langle E_i \rangle$ for $i = 1, 2$ is the intersection of the span $\langle E_i \rangle$ with the boundary of \bar{M}_x , and where g is the projective rank of the guts $\langle E_1 \rangle \cap \langle E_2 \rangle$ of the subseparation (E_1, E_2) . Clearly, one may compute boundary data of $\bar{M}_x = \bar{P}(T_x)$ from boundary data of $\bar{P}(T'_x)$ and $\bar{P}(T''_x)$, and from the composition operator labeling x in (parametrized) constant time. Hence the whole algorithm is implemented in linear time. \square

ALGORITHM 4.10 (computing a good partition (F_1, F_2) for Lemma 4.8).

Input: A $\leq t'$ -boundaried parse tree T parsing the represented matroid $N = P(T)$, and a subset $F_0 \subseteq E(N)$ such that $\lambda_N(F_0) = 3t$.

Output: A partition (F_1, F_2) of the set $E(N) - F_0$ such that $\lambda_N(F_1), \lambda_N(F_2) \leq 3t$ (or an answer NO if no such partition exists) computed in time $O(|V(T)|)$.

Proof. This is a straightforward extension of the dynamic program implemented in Algorithm 4.9. Moreover, for each instance of boundary data in this case, we record one representative of the partition we compute. Since the ground set of N can be easily implemented so that the operation of a set union (of the representatives) takes constant time, the overall computing time is still linear in T . We leave details to the reader. \square

Now we come to the hard part—computing a valid composition operator for the new node of the parse tree T' in step 3(b) or 3(c). We (implicitly) know the three boundaries of the composition operator from a defining tripartition of the matroid elements. However, we have to choose independent spanning sets of generator points for the boundaries in such a way that the two matching boundaries of adjacent composition operators in the parse tree T' get the same generator points. For a parse tree U , we call a *virtual point* of U any point which is contained in the span of some composition operator in the tree U . The concept of virtual points of the parse tree U allows us to determine relative positions of certain points with respect to the elements of the matroid $P(U)$, without the necessity of using absolute vectors in a particular matroid representation.

If (F_1, F_2) is a separation in the represented matroid $P(U)$ parsed by U , then we express spanning generator points for the guts of (F_1, F_2) as a sequence of virtual points of the parse tree U . We naturally say that a set Z of virtual points in U is independent if Z is linearly independent in the point configuration parsed by U . Next follows the corresponding extension of Algorithm 4.9: The fact that the guts are always spanned by some virtual points of U is implicitly proved in the algorithm.

ALGORITHM 4.11 (computing virtual points spanning the guts of a given separation over a parse tree).

Input: An $\leq t'$ -boundaried parse tree T parsing the matroid $N = P(T)$, and a partition (F_1, F_2) of $E(N)$ such that $\lambda_N(F_1) \leq 3t$. (The partition is symmetric, i.e.,

(F_1, F_2) is considered the same as (F_2, F_1) .)

Output: A uniquely determined sequence $\{x_1, x_2, \dots, x_k\}$, $k = \lambda_N(F_1) - 1$, of independent virtual points of the parse tree T which span the guts of the separation (F_1, F_2) . This is computed in time $O(|V(T)|)$.

Proof. Let x be a node of T labeled by \odot , and let T_x be the rooted subtree of T with the root x . Denote by $\bar{M}_x = \bar{P}(T_x)$ and by T'_x, T''_x the left and right rooted subtrees of x in T . Analogously to Algorithm 4.9, we call *boundary data* of \bar{M}_x the quadruple $(\Sigma_1, \Sigma_2, g, Z)$, where $E_i = F_i \cap J(\bar{M}_x)$ and $\Sigma_i = \partial(\bar{M}_x) \cap \langle E_i \rangle$ for $i = 1, 2$, where $g = r(\langle E_1 \rangle \cap \langle E_2 \rangle)$, and where $Z = \{z_1, \dots, z_g\}$ is a sequence of independent virtual points of T which span the guts of (E_1, E_2) in \bar{M}_x . Moreover, we require that the points of $Z \cap \partial(\bar{M}_x)$ contained in the boundary span the set $\Sigma_1 \cap \Sigma_2$ (boundary-span condition).

We denote $\bar{M}'_x = \bar{P}(T'_x)$, $\bar{M}''_x = \bar{P}(T''_x)$ and $E'_i = F_i \cap J(\bar{M}'_x)$, $E''_i = F_i \cap J(\bar{M}''_x)$. The guts $\langle E_1 \rangle \cap \langle E_2 \rangle$ are clearly spanned by the four sets $\langle E'_1 \rangle \cap \langle E'_2 \rangle$, $\langle E''_1 \rangle \cap \langle E''_2 \rangle$, $\langle E'_1 \rangle \cap \langle E''_2 \rangle$, $\langle E''_1 \rangle \cap \langle E'_2 \rangle$. The latter two sets $\langle E'_1 \rangle \cap \langle E''_2 \rangle$ and $\langle E''_1 \rangle \cap \langle E'_2 \rangle$ are contained in the span of the composition operator \odot labeling x in the parse tree T by definition. Hence, using the boundary-span condition above, we can compute boundary data \bar{M}_x from boundary data of \bar{M}'_x and \bar{M}''_x , and from the composition operator \odot in a canonical way.

Since the guts of (E_1, E_2) have bounded rank $g \leq 3t$ in \bar{M}_x , boundary data carry only a limited amount of information. One node x of the parse tree T is processed in time depending on \mathbb{F} and t , but not depending on the size of T . So the whole algorithm is implemented in (parametrized) linear time. \square

Lastly, we present an analogue of Algorithm 4.6 computing the composition operator from sequences of virtual points. Consider three sequences Z_1, Z_2, Z_3 of virtual points in a common parse tree T , each Z_i , $i = 1, 2, 3$, formed by g_i independent points. Then these sequences represent the composition operator $(Z, \gamma_1, \gamma_2, \gamma_3)$, where $Z = Z_1 \cup Z_2 \cup Z_3$ is the ground point configuration as parsed by T , and γ_i maps the elements of the sequence in order $Z_i = \{\gamma_i(1), \gamma_i(2), \dots, \gamma_i(g_i)\}$ for $i = 1, 2, 3$.

ALGORITHM 4.12 (computing the composition operator from the given virtual points in a parse tree).

Input: Three independent sequences Z_1, Z_2, Z_3 of virtual points in a common $\leq t'$ -boundaried parse tree T , where each sequence has at most $3t$ members.

Output: The $\leq 3t$ -boundaried composition operator represented by the virtual points of Z_1, Z_2, Z_3 , computed in time $O(|V(T)|)$.

Proof. We argue similarly as in Algorithm 4.6. There is a bounded number of $\leq t'$ -boundaried composition operators, and each one of them may be identified by a bounded number of linear vector equations and inequations. Moreover, these equations are homogeneous, and thus invariant under nonsingular vector transformations by the definition of a composition operator. Thus we can decide their validity for the input from information given in the parse tree T .

Let x be a node of T labeled by \odot^x , and let T_x be the rooted subtree of T with the root x . Denote by T'_x, T''_x the left and right rooted subtrees of x in T by $\bar{M}_x = \bar{P}(T_x)$ and by $Z_x \subseteq Z_1 \cup Z_2 \cup Z_3$ those of given virtual points of T which are contained in the composition operators in the subtree T_x . We use the following dynamic program: We call *boundary-combination data* of \bar{M}_x the list of all linear combinations of the virtual points Z_x which result in a point in the boundary $\partial(\bar{M}_x)$. This is a well-defined notion according to the definition of a parse tree and to elementary linear algebra, and one can determine boundary-combination data of $\bar{M}_x = \bar{P}(T_x)$ from boundary-

combination data of $\bar{P}(T'_x)$ and of $\bar{P}(T''_x)$, and from the composition operator \odot^x labeling x . Since boundary-combination data carry a limited amount of information at each tree node, the whole parse tree T can be processed in linear parametrized time.

Consider a homogeneous linear (in)equation EQ over the points $Z_1 \cup Z_2 \cup Z_3$. Then, obviously, we can decide validity of EQ from boundary-combination data at the node y of T in which the last point involved in EQ is encountered in the set Z_y . This takes constant time. Therefore, for any $\leq 3t$ -boundaried composition operator \odot , we can decide whether the virtual points of Z_1, Z_2, Z_3 represent \odot . In this way we find the $\leq 3t$ -boundaried composition operator represented by Z_1, Z_2, Z_3 in total linear (parametrized) time $O(|V(T)|)$. \square

LEMMA 4.13. *Step 4.1 in Algorithm 4.1 is correctly implemented in time $O(n^2)$ for fixed parameters $t, t' \leq 10t$, and \mathbb{F} .*

Proof. Let $M = M(\mathbf{A})$ be the given matroid on n elements, and let the set X and the $\leq t'$ -boundaried parse tree T be as in Algorithm 4.1. (So $P(T) = M \upharpoonright X$.) Notice that the size of the partial parse trees considered in the algorithm is at most linear in $|X| \leq n$. The important point of the implementation of step 4.1 in Algorithm 4.1 is that the boundaried matroids labeling the partial spanning tree T' are not explicitly described: Instead of a boundaried matroid \bar{N} , we record only the set $J(\bar{N})$ of its internal elements. The boundary subspace of \bar{N} is then implicitly given by the guts of the separation $(J(\bar{N}), X - J(\bar{N}))$, and the boundary points are handled as virtual points in the parse tree T .

Part 3(a) of Algorithm 4.1 is trivial to implement. Let us look at part 3(b): The boundary rank of the boundaried matroid \bar{N} labeling a chosen leaf ℓ of T' is computed in time $O(n)$ by Algorithm 4.9. Suppose that the computed rank is less than $3t$. Let $e \in J(\bar{N})$ be an arbitrary element. Then the new composition operator being added to the parse tree T' corresponds to a tripartition $X_1 = J(\bar{N}) - \{e\}$, $X_2 = \{e\}$, and $X_3 = X - J(\bar{N})$. Rest is described below.

Suppose that the above computed boundary rank of \bar{N} equals $3t$. Then we are in part 3(c): We call Algorithm 4.10 for $F_0 = X_3 = X - J(\bar{N})$ over the parse tree T . The result is a partition (X_1, X_2) of the set $X - X_3 = J(\bar{N})$, which does exist by Lemma 4.8 if branch-width of M is at most $t + 1$. The whole tripartition of X to X_1, X_2, X_3 then determines the new composition operator and the new leaf labels which will be added to T' . On the other hand, if branch-width of M exceeds $t + 1$, then this part may possibly fail, and then Algorithm 4.1 ends without a parse tree and with an error message. Computing time is $O(n)$ here.

So far, we have constructed a tripartition (X_1, X_2, X_3) of the set X such that $\lambda_{M \upharpoonright X}(X_i) \leq 3t$ for $i = 1, 2, 3$, and we are going to find the composition operator that would “glue” these three parts together in an enlarged parse tree T'_1 : We call Algorithm 4.11 for the separation $(X_i, X - X_i)$ over the parse tree T , for each $i = 1, 2, 3$. Then we call Algorithm 4.12 for the resulting three sequences $Z_i, i = 1, 2, 3$, of virtual points over the parse tree T to construct a composition operator \odot' . We construct the new tree T'_1 from T' by adding two new leaves ℓ_1, ℓ_2 as sons of ℓ . We label $\ell_i, i = 1, 2$, by the boundaried matroid \bar{N}_i induced on the elements X_i and the boundary Z_i , and we relabel ℓ with \odot' . (Rest of T'_1 is unchanged.) This is all done in time $O(n)$ again.

It remains to prove correctness of the above construction by induction on $|V(T')|$. At the beginning, when T' has one node, the matroid \bar{N} has boundary rank 0, and $X_3 = \emptyset$, so Z_3 is an empty sequence. Otherwise, the sequence Z_3 coincides with the set

of boundary points of \bar{N} since they were both determined uniquely by Algorithm 4.11 for the same separation $(X_3, J(\bar{N}))$. Therefore, $\bar{N}_1 \odot' \bar{N}_2 = \bar{N}$, and so $\bar{P}(T'_1) = \bar{P}(T')$ and $P(T'_1) = P(T) = M \upharpoonright X$.

Since each iteration of step 4.1 adds a new leaf to the constructed parse tree T' , there are at most $|X| - 1 = O(n)$ iterations. Thus step 4.1 is finished in time $O(n^2)$. We remark that our implementation works for an arbitrary finite field \mathbb{F} and an integer $t \geq 1$ given as parameters. \square

4.3. Conclusion: Implementation of Algorithm 4.1. Using Lemmas 4.7 and 4.13, and the preceding description, we immediately conclude as follows.

THEOREM 4.14. *Let us fix an integer $t \geq 1$ and a finite field \mathbb{F} , and consider a given n -element \mathbb{F} -represented matroid $M = M(\mathbf{A})$.*

(a) *If branch-width of M is at most $t+1$, then Algorithm 4.1 computes a spanning $\leq 3t$ -boundaried parse tree T for M in time $O(n^3)$.*

(b) *If branch-width of M exceeds $t+1$, then Algorithm 4.1 (possibly) ends with no output parse tree, but the computation is also finished in time $O(n^3)$.* \square

5. Parametrized complexity of branch-width. There are many connections between tree-width of graphs and parametrized complexity of hard graph problems [9, Chapter 6]. A large class of natural graph problems, including the notoriously hard like Hamiltonicity or 3-coloring problems, can be expressed in monadic second-order logic. By the results of Courcelle [6, 7], and Arnborg, Lagergren, and Seese[2], all such monadic second-order (MSO)-definable problems can be solved quickly for (incidence) graphs with a tree-decomposition of bounded width. A similar phenomenon occurs for represented matroids, and we can use that to determine the exact value of branch-width of a represented matroid, in addition to an approximation following from Algorithm 4.1.

5.1. MSO logic of matroids. The *monadic second-order logic (MSOL) of matroids* is defined as follows: The syntax includes variables for matroid elements and element sets, the quantifiers \forall, \exists applicable to these variables, the logical connectives \wedge, \vee, \neg , and the following predicates:

1. $=$, the equality for elements and their sets;
2. $e \in F$, where e is an element and F is an element set variable;
3. $\text{indep}(F)$, where F is an element set variable, and the predicate tells whether F is independent in the matroid.

In our paper, we follow a tree-automata formalization of Courcelle’s result, as in [1].

THEOREM 5.1 (Hliněný [14]). *Let $t \geq 1$, and let \mathbb{F} be a finite field. Assume \mathcal{M} is a set of represented matroids over \mathbb{F} described by a sentence in the MSOL of matroids. Then there is a finite tree automaton accepting exactly the $\leq (t - 1)$ -boundaried parse trees of members of \mathcal{M} (of branch-width bounded by t).*

We remark that the proof [14] of Theorem 5.1 is constructive—there is an algorithm that computes the accepting tree automaton for the given field \mathbb{F} , the formula ϕ describing \mathcal{M} , and t .

5.2. Parametrized complexity. When speaking about parametrized complexity, we closely follow [9]. Here we present only the basic definition of parametrized tractability. For simplicity, we restrict the definition to decision problems, although an extension to computation problems is straightforward. Let Σ be the input alphabet. A *parametrized problem* is an arbitrary subset $A^p \subseteq \Sigma^* \times \mathbb{N}$. For an instance $(x, k) \in A^p$, we call k the *parameter* and x the input for the problem. (The parameter

is sometimes implicit in the context.)

DEFINITION 5.2. We say that a parametrized problem A^p is (nonuniformly) fixed-parameter tractable if there is a sequence of algorithms $\{\mathcal{A}_i : i \in \mathbb{N}\}$, and a constant c , such that $(x, k) \in A^p$ if and only if the algorithm \mathcal{A}_k accepts (x, k) , and if the running time of \mathcal{A}_k on (x, k) is $O(|x|^c)$ for each k .

We say that a parametrized problem A^p is uniformly fixed-parameter tractable if there is an algorithm \mathcal{A} , a constant c , and an arbitrary function $f : \mathbb{N} \rightarrow \mathbb{N}$ such that $(x, k) \in A^p$ if and only if the algorithm \mathcal{A} accepts (x, k) , and if the running time of \mathcal{A} on (x, k) is $O(f(k) \cdot |x|^c)$.

In our context, the input X is an \mathbb{F} -represented matroid, and the parameter k is an upper bound on the branch-width of X . (Notice that correctness of the assumed branch-width bound is implicitly checked in Algorithm 4.1.) In a more general setting, we may even consider the parameter as a pair (\mathbb{F}, k) encoded as an integer.

5.3. Computing branch-width exactly. We use Theorem 5.1 to determine branch-width in the following way. We remark that, unlike for graph minors, it is not known how to test for a fixed matroid minor in polynomial time.

LEMMA 5.3. For every matroid N there is an MSOL formula ψ_N such that $\psi_N \models M$ (i.e., ψ_N is true on a matroid M) if and only if N is a minor of M .

Proof. We include a short proof here; more details can be found in [15]. Matroid N is a minor of M if there are two sets C, D such that $N = M \setminus D/C$. Suppose that $N = M \setminus D/C$ holds. Then a set $X \subseteq E(N)$ is dependent in N if and only if there is a dependent set $Y \subseteq E(M)$ in M such that $Y - X \subseteq C$.

Since N is fixed, we may identify the elements of an N -minor in M by variables x_1, \dots, x_n in order, where $n = |E(N)|$. For each $J \subseteq [1, n]$, we write

$$\text{mdep}(x_j : j \in J; C) \equiv \exists Y \left[\neg \text{indep}(Y) \wedge \forall y \left(y \notin Y \vee y \in C \vee \bigvee_{j \in J} y = x_j \right) \right].$$

Now, $M \setminus D/C$ is isomorphic to N if and only if the dependent subsets of $\{x_1, \dots, x_n\}$ exactly match the dependent sets of N . Hence we express ψ_N as

$$\psi_N \equiv \exists C \exists x_1, \dots, x_n \left[\bigwedge_{1 \leq i < j \leq n} x_i \neq x_j \wedge \bigwedge_{J \in \mathcal{J}_+} \neg \text{mdep}(x_j : j \in J; C) \wedge \bigwedge_{J \in \mathcal{J}_-} \text{mdep}(x_j : j \in J; C) \right],$$

where \mathcal{J}_+ is the set of all $J \subseteq [1, n]$ such that $\{x_j : j \in J\}$ actually is independent in N , and where \mathcal{J}_- is the complement of \mathcal{J}_+ . \square

The class \mathcal{B}_k of matroids of branch-width at most k is closed under taking minors, and so membership in \mathcal{B}_k can be tested by looking for the excluded (or forbidden) minors for \mathcal{B}_k . By the result of [11], the excluded minors for the class \mathcal{B}_k have size at most $(6^{k+1} - 1)/5$, and hence their number is finite and they can all be found by a brute force algorithmic search.

COROLLARY 5.4. For every $k \geq 1$, there is a computable MSOL formula ϕ_k such that $\phi_k \models M$ if and only if M has branch-width at most k (i.e., shortly $\phi_k(M) \equiv M \in \mathcal{B}_k$). \square

THEOREM 5.5. Let \mathbb{F} be a finite field, and let $t \geq 1$ be a constant. There is an algorithm that, given a rank r matrix $\mathbf{A} \in \mathbb{F}^{r \times n}$ such that the branch-width of the matroid $M(\mathbf{A})$ is at most $t + 1$, finds the exact branch-width of $M(\mathbf{A})$ in time $O(n^3)$.

Proof. For $k = 2, 3, \dots, t + 1$, the algorithm first precomputes all the excluded minors for the class \mathcal{B}_k , and the formulas ϕ_k from Corollary 5.4. Then the algorithm

computes the finite tree automaton \mathcal{A}_k from Theorem 5.1, which accepts the parse trees for represented matroids described by ϕ_k . (This precomputation is done in time not depending on M .)

Given an n -element matroid $M(\mathbf{A})$ represented over \mathbb{F} , the algorithm calls Algorithm 4.1 to produce an $\leq 3t$ -boundaried parse tree T of $M(\mathbf{A})$ in time $O(n^3)$. Then it finds the smallest $k_0 \leq t + 1$ such that the parse tree T is accepted by \mathcal{A}_{k_0} , where each automaton \mathcal{A}_k is emulated in time $O(n)$. The branch-width of $M(\mathbf{A})$ is k_0 . \square

COROLLARY 5.6. *For a finite field \mathbb{F} , the branch-width of an \mathbb{F} -represented matroid is a uniformly fixed-parameter tractable problem.* \square

Remark 5.7. We may analogously argue in the case of matroid tree-width, which has been defined in [16]: The tree-width of a matroid represented over a finite field is nonuniformly fixed-parameter tractable. However, we do not have a size-bound analogous to [11] at hand, and so we have to use a nonconstructive well-quasi-ordering argument of [10] to establish existence of a finite list of excluded minors for represented matroids of tree-width at most k . Since the definition of matroid tree-width is not easy, we include no formal statements here.

6. Concluding remarks. Using Theorem 4.14, one may easily derive the following corollary of Theorem 5.1.

COROLLARY 6.1. *Let $t \geq 1$, let \mathbb{F} be a finite field, and let ϕ be a sentence in the MSOL of matroids. Consider a given n -element \mathbb{F} -represented matroid $M = M(\mathbf{A})$ of branch-width at most t . The question of whether ϕ is true for the matroid $M(\mathbf{A})$ is uniformly fixed-parameter tractable with respect to the combined parameter (\mathbb{F}, t, ϕ) . If \mathbb{F} , t , and ϕ are fixed, then the answer can be computed from the matrix \mathbf{A} in time $O(n^3)$.*

More similar algorithmic applications, including recognition of any minor-closed matroid family, can be found in [15]. Besides applications based directly on Theorem 5.1, we may use the machinery of matroid parse trees from sections 3 and 4 for solving other problems. For example, we provide a straightforward recursive formula and an algorithm for computing the Tutte polynomial of a represented matroid in [13].

THEOREM 6.2 (Hliněný [13]). *Let $t \geq 1$, and let \mathbb{F} be a finite field. Consider a given n -element \mathbb{F} -represented matroid $M = M(\mathbf{A})$ of branch-width at most t . Then the Tutte polynomial $T(M(\mathbf{A}); x, y)$ of $M(\mathbf{A})$ can be computed in time $O(n^6 \log n \log \log n)$.*

Moreover, a recent research of Oum shows that our matroid results are of interest also in graph theory—he uses Algorithm 4.1 to approximate the clique-width of a graph in parametrized cubic time [19]. That application uses important notion of rank-width [18], defined by the matrix rank function on adjacency matrices of graphs. (No efficient approximation algorithms for graph clique-width were known before the introduction of rank-width.) In particular, the clique-width of a graph of rank-width r is between r and $2^{r+1} - 1$, and the rank-width of a bipartite graph G equals the branch-width minus one of the matroids represented over $GF(2)$ by the bipartite adjacency matrix of G . Oum’s result is briefly stated as follows.

THEOREM 6.3 (Oum [19]). *For every fixed $r > 0$, there is an algorithm checking whether the rank-width of a given graph G is at most r in time $O(n^3)$. Moreover, the algorithm outputs a rank-decomposition of width at most $24r$ in the “yes” case.*

It is interesting to watch the radical structural change when we move from represented matroids over finite fields to general abstract matroids, or even to matroids over infinite fields. For example, Theorem 5.1 [14] is provably false even for matroids that are represented over the integers by matrices with entries from $\{-1, 1, 3\}$.

Also, the problems of Corollary 6.1 and Theorem 6.2 become NP-hard for matroids of branch-width 3 over the integers. The computational borderline is even more clear when considering decidability of matroid theories [17]: While MSO theories of the matroids of bounded branch-width are decidable over finite fields (and, conversely, decidability of such a theory implies a bound on branch-width), the MSO theory of all matroids of branch-width 3 is undecidable.

The situation seems to be slightly different for the problem of branch-width itself, at least when branch-width is 3. We present in [12] an easy algorithm that decides whether a matroid has branch-width at most 3 in polynomial time. (The algorithm also has a fast practical implementation.) This algorithm is not restricted to represented matroids—it works for all matroids for which the rank function can be efficiently determined. Unfortunately, there seems to be no straightforward way to extend the algorithm to higher values of branch-width.

PROBLEM 6.4. *What is the parametrized complexity of the problem to determine the branch-width of a matroid M ?*

- (a) *If $M = M(\mathbf{A})$ is given by a matrix representation over an infinite field?*
- (b) *If M is given by a rank oracle?*

Acknowledgments. The author would like to thank Geoff Whittle for valuable discussions about matroid representations and branch-width, and to thank the anonymous referees for helpful suggestions to improve the paper presentation.

REFERENCES

- [1] K. A. ABRAHAMSON AND M. R. FELLOWS, *Finite automata, bounded treewidth, and well-quasiordering*, in Graph Structure Theory, Contemp. Math. 147, AMS, Providence, RI, 1993, pp. 539–564.
- [2] S. ARNBORG, J. LAGERGREN, AND D. SEESE, *Easy problems for tree-decomposable graphs*, J. Algorithms, 12 (1991), pp. 308–340.
- [3] H. L. BODLAENDER, *A tourist guide through treewidth*, Acta Cybernet., 11 (1993), pp. 1–21.
- [4] H. L. BODLAENDER, *A linear-time algorithm for finding tree-decompositions of small treewidth*, SIAM J. Comput., 25 (1996), pp. 1305–1317.
- [5] H. L. BODLAENDER AND D. M. THILIKOS, *Constructive linear time algorithms for branchwidth*, in Proceedings of the 24th ICALP, Lecture Notes in Comput. Sci. 1256, Springer, Berlin, 1997, pp. 627–637.
- [6] B. COURCELLE, *On context-free sets of graphs and their monadic second-order theory*, in Graph Grammars and Their Application to Computer Science, Lecture Notes in Comput. Sci. 291, Springer, Berlin, 1987, pp. 133–146.
- [7] B. COURCELLE, *The monadic second-order logic of graphs I. Recognizable sets of finite graphs*, Inform. Comput., 85 (1990), pp. 12–75.
- [8] R. DIESTEL, *Graph Theory*, 2nd ed., Graduate Texts in Math. 173, Springer-Verlag, New York, 2000.
- [9] R. G. DOWNEY AND M. R. FELLOWS, *Parametrized Complexity*, Springer-Verlag, New York, 1999.
- [10] J. F. GEELLEN, A. H. M. GERARDS, AND G. P. WHITTLE, *Branch-width and well-quasi-ordering in matroids and graphs*, J. Combin. Theory Ser. B, 84 (2002), pp. 270–290.
- [11] J. F. GEELLEN, A. H. M. GERARDS, N. ROBERTSON, AND G. P. WHITTLE, *On the excluded minors for the matroids of branch-width k* , J. Combin. Theory Ser. B, 88 (2003), pp. 261–265.
- [12] P. HLINĚNÝ, *On the excluded minors for matroids of branch-width three*, Electron. J. Combin., 9 (2002), Research paper 32. Available online at <http://www.combinatorics.org>.
- [13] P. HLINĚNÝ, *The Tutte polynomial for matroids of bounded branch-width*, Combin. Probab. Comput., to appear.
- [14] P. HLINĚNÝ, *Branch-width, parse trees, and monadic second-order logic for matroids*, J. Combin. Theory Ser. B, submitted.
- [15] P. HLINĚNÝ, *On matroid properties definable in the MSO logic*, in Proceedings of the Mathematical Foundations of Computer Science (MFCS 2003), Lecture Notes in Comput. Sci. 2747,

- Springer-Verlag, Berlin, 2003, pp. 470–479.
- [16] P. HLINĚNÝ AND G. P. WHITTLE, *Matroid tree-width*, European J. Combin., submitted.
 - [17] P. HLINĚNÝ AND D. SEESE, *Trees, grids, and MSO decidability: From graphs to matroids*, Theoret. Comput. Sci., to appear.
 - [18] SANG-IL OUM AND P. D. SEYMOUR, *Approximating clique-width and branch-width*, submitted.
 - [19] SANG-IL OUM, *Approximating rank-width and clique-width quickly*, in Proceedings of the 31st International Workshop on Graph-Theoretic Concepts in Computer Science, Springer-Verlag, Heidelberg, to appear.
 - [20] J. G. OXLEY, *Matroid Theory*, Oxford University Press, Oxford, UK, 1992.
 - [21] N. ROBERTSON AND P. D. SEYMOUR, *Graph minors—A survey*, in Surveys in Combinatorics, Cambridge University Press, Cambridge, UK, 1985, pp. 153–171.
 - [22] N. ROBERTSON AND P. D. SEYMOUR, *Graph minors x. Obstructions to tree-decomposition*, J. Combin. Theory Ser. B, 52 (1991), pp. 153–190.

CORRECTIONS TO “A PARAMETRIZED ALGORITHM FOR MATROID BRANCH-WIDTH”

Due to a TeX error, the steps of Algorithm 4.1 are cited incorrectly on pages 267, 268, 269, 272, and 273 of “A Parametrized Algorithm for Matroid Branch-Width,” *SIAM Journal on Computing*, 35 (2005), pp. 259–276, by Petr Hliněný. They should read as follows:

Page 267, line 8: Step 2.
Page 267, line 9: Step 3.
Page 267, line 19: Step 2; Step 1.
Page 267, line 20: Step 2.

Page 268, line 41: Step 2.
Page 268, line 43: Step 2.

Page 269, line 10: Step 3.
Page 269, line 11: Step 3.
Page 269, line 44: Step 2.

Page 272, line 12: Step 3.
Page 272, line 17: Step 3.

Page 273, line 4: Step 3.

SIAM regrets this error.

ON THE PERFORMANCE OF GREEDY ALGORITHMS IN PACKET BUFFERING*

SUSANNE ALBERS[†] AND MARKUS SCHMIDT[†]

Abstract. We study a basic buffer management problem that arises in network switches. Consider m input ports, each of which is equipped with a buffer (queue) of limited capacity. Data packets arrive online and can be stored in the buffers if space permits; otherwise packet loss occurs. In each time step the switch can transmit one packet from one of the buffers to the output port. The goal is to maximize the number of transmitted packets. Simple arguments show that any work-conserving algorithm, which serves any nonempty buffer, is 2-competitive. Azar and Richter recently presented a randomized online algorithm and gave lower bounds for deterministic and randomized strategies.

In practice, greedy algorithms are very important because they are fast, use little extra memory, and reduce packet loss by always serving a longest queue. In this paper we first settle the competitive performance of the entire family of greedy strategies. We prove that greedy algorithms are not better than 2-competitive no matter how ties are broken. Our lower bound proof uses a new recursive construction for building adversarial buffer configurations that may be of independent interest. We also give improved lower bounds for deterministic and randomized online algorithms.

In this paper we present the first deterministic online algorithm that is better than 2-competitive. We develop a modified greedy algorithm, called *semigreedy*, and prove that it achieves a competitive ratio of $17/9 \approx 1.89$. The new algorithm is simple, fast, and uses little extra memory. Only when the risk of packet loss is low does it not serve the longest queue. Additionally we study scenarios when an online algorithm is granted additional resources. We consider resource augmentation with respect to memory and speed; i.e., an online algorithm may be given larger buffers or higher transmission rates. We analyze greedy and other online strategies.

Key words. buffer, competitive, greedy, network switch, online, packet, throughput

AMS subject classifications. 90B35, 90B36, 68M20, 68Q25, 68W01

DOI. 10.1137/S0097539704446268

1. Introduction. The performance of high-speed networks critically depends on switches that route data packets arriving at the input ports to the appropriate output ports so that the packets can reach their correct destinations in the network. To reduce packet loss when the traffic is bursty, ports are equipped with buffers in which packets can be stored temporarily. However, the buffers are of limited capacity so that effective buffer management strategies are important for maximizing the throughput at a switch. As a result there has recently been considerable research interest in the design and analysis of various buffer management policies [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16].

We study a very basic problem in this context. Consider m input ports which serve a given output port. Each input port has a buffer that can simultaneously store up to B packets and is organized as a queue. In any time step new packets may arrive at the input ports and can be appended to the corresponding buffers if space permits. More specifically, suppose that the buffer at port i currently stores b_i packets and

*Received by the editors October 1, 2004; accepted for publication (in revised form) June 9, 2005; published electronically October 7, 2005. A preliminary version of this paper appeared in *Proceedings of the 36th Annual ACM Symposium on Theory of Computing (STOC '04)*, ACM, New York, 2004, pp. 35–44.

<http://www.siam.org/journals/sicomp/35-2/44626.html>

[†]Institut für Informatik, Albert-Ludwigs-Universität Freiburg, Georges-Köhler-Allee 79, 79110 Freiburg, Germany (salbers@informatik.uni-freiburg.de, markus.schmidt@informatik.uni-freiburg.de). The work of the first author was supported by Deutsche Forschungsgemeinschaft project AL 464/4-1 and by EU projects APPOL and APPOL II.

that a_i new packets arrive there. If $b_i + a_i \leq B$, then all new packets can be accepted; otherwise $a_i + b_i - B$ packets must be dropped. In any time step the switch can select one nonempty buffer and transmit the packet at the head through the output port. We assume w.l.o.g. that the packet arrival step precedes the transmission step. The goal is to maximize the *throughput*, i.e., the total number of transmitted packets. The scenario we study here arises, for instance, in input-queued (IQ) switches, which represent the dominant switch architecture today. In an IQ switch with m input and m output ports, packets that arrive at input i and have to be routed to output j are buffered in a virtual output queue Q_{ij} . In each time step, for any output j , one data packet from queues Q_{ij} , $1 \leq i \leq m$, can be sent to that output. The buffer size B is large—typically several hundreds or thousands. We emphasize that we consider all packets to be equally important, i.e., they all have the same value. Most current networks, in particular IP networks, treat packets from different data streams equally in intermediate switches.

Information on future packet arrivals usually is very limited or not available at all. We make no probabilistic assumptions about the input and investigate an online setting, where at any time future packet arrivals are unknown. We are interested in online buffer management strategies that have a provably good performance. Following [17] we call a deterministic online algorithm ALG c -competitive if $c \cdot T_{ALG}(\sigma) \geq T_{OPT}(\sigma)$ for all packet arrival sequences σ . Here $T_{ALG}(\sigma)$ and $T_{OPT}(\sigma)$ denote the throughputs achieved by ALG and by an optimal offline algorithm OPT that knows the entire input σ in advance. If ALG is a randomized algorithm, then $T_{ALG}(\sigma)$ has to be replaced by the expected throughput $E[T_{ALG}(\sigma)]$.

In practice, greedy algorithms are most important. At any time, a greedy algorithm serves a queue that currently buffers the largest number of packets. Serving the longest queue is a very reasonable strategy to avoid packet loss if future arrival patterns are unknown. Moreover, greedy strategies are interesting because they are fast and use little extra memory. A switch cannot afford complex computations to decide which queue to serve, nor has it sufficient memory to maintain detailed information on past or current configurations. In this paper we present a thorough study of greedy algorithms and their variants.

Previous work. The following simple observation shows that any work-conserving algorithm ALG , which serves any nonempty queue, is 2-competitive: Partition σ into subsequences σ_l such that ALG 's buffers are empty at the end of each σ_l . W.l.o.g. we postpone the beginning of σ_{l+1} until OPT has emptied its buffers, too. If OPT buffers b_i packets in queue i at the end of subsequence σ_l , then at least b_i packets must have arrived there in σ_l . ALG has transmitted at least $\sum_{i=1}^m b_i$ packets because it has accepted at least $\sum_{i=1}^m b_i$ packets and all its buffers are empty again. Since both ALG and OPT transmit exactly one packet during each time step and OPT still buffers $\sum_{i=1}^m b_i$ packets when ALG 's buffers become empty, OPT delivers $\sum_{i=1}^m b_i$ more than ALG does.

Prior to our work, no deterministic online algorithm with a competitive ratio smaller than 2 was known. Azar and Richter [4] showed that if $B = 1$, no deterministic strategy can be better than $(2 - \frac{1}{m})$ -competitive. For arbitrary B , they gave a lower bound of 1.366. Azar and Richter also considered randomized algorithms and presented a strategy that achieves a competitiveness of $e/(e-1) \approx 1.58$. For $B = 1$, they showed a lower bound of 1.46 on the performance of any randomized online algorithm.

Bar-Noy et al. [9] and Fleischer and Koga [10] studied buffer management policies when buffers have unlimited capacity and one wishes to minimize the maximum queue

length. They presented $\Theta(\log m)$ -competitive online algorithms. Additional results are known when packets have values and the goal is to maximize the total value of the transmitted packets. Almost all of the previous work has focused on the single queue problem; i.e., we have to maintain only one buffer. Kesselman et al. [12] gave 2-competitive algorithms for various models where preemption is allowed; i.e., packets admitted to the queue may be discarded in the event of buffer overflow. Recently, Kesselman, Mansour, and van Stee [14] developed a 1.983-competitive algorithm when packets must be transmitted in the order they arrive. The bound was improved to 1.75 by Bansal et al. [8]. Aiello et al. [1] investigated single queue problems assuming that preemption is not allowed. For this scenario Andelman, Mansour, and Zhu [2] showed tight bounds of $\Theta(\log \alpha)$, where α is the ratio of the maximum to the minimum packet value.

Azar and Richter [4] presented a technique that transforms any c -competitive algorithm for a single queue into a $2c$ -competitive algorithm for m queues. Using results from [2, 12] they derived 4-competitive preemptive and $2e\lceil \ln \alpha \rceil$ -competitive nonpreemptive algorithms. An improved 3-competitive preemptive algorithm was given in [5].

Our contribution. In the first part of the paper we settle the competitive performance of the entire family of greedy algorithms. We prove that a greedy algorithm cannot be better than 2-competitive, no matter how ties are broken. Since any work-conserving algorithm is 2-competitive, the competitiveness of any greedy policy is indeed 2. Our lower bound construction is involved and relies on a new recursive construction for building dynamic adversarial buffer configurations. We believe that our technique may be useful for proving lower bounds in other multiqueue buffering problems. In fact, we use a variant of our technique to develop a lower bound for any deterministic online algorithm. We show that, for any buffer size B , no deterministic online strategy ALG can achieve a competitiveness smaller than $e/(e-1) \approx 1.58$. Interestingly, we establish this bound by comparing the throughput of ALG to that of any greedy algorithm. Using an approach different from [4], we show that for any B , a randomized online algorithm cannot be better than 1.46-competitive, which is exactly the same bound as the one shown in [4] for $B = 1$. Since the techniques used in [4] cannot be generalized to $B > 1$, our method is independent of previous work.

Although in terms of competitiveness greedy algorithms are not better than arbitrary work-conserving algorithms, greedy strategies are important from a practical point of view. Therefore it is interesting to consider variants of greedy policies and to analyze greedy approaches in extended problem settings. In the second part of the paper we develop a slightly modified deterministic greedy strategy, called *semigreedy* (SGR), and prove that it achieves a competitive ratio of $17/9 \approx 1.89$. We conjecture that SGR is actually an optimal deterministic algorithm because, for $B = 2$, we give a proof that it achieves an optimal competitiveness of $13/7 \approx 1.86$. These results show, in particular, that deterministic algorithms can beat the factor of 2 and perform better than arbitrary work-conserving strategies.

The new SGR algorithm is simple. If there is a queue buffering more than $\lfloor B/2 \rfloor$ packets, SGR serves a longest queue. If all queues store at most $\lfloor B/2 \rfloor$ packets, SGR serves a longest queue that has never buffered B packets, provided there is one; otherwise SGR serves a longest queue. The idea of this rule is to establish some fairness among the queues. SGR is essentially as fast as greedy. It can be implemented such that at most one extra comparison is needed in each time step. The extra memory requirements are also low. For each queue, we have only to maintain one

bit indicating whether or not the queue has ever buffered B packets. SGR does not follow the standard greedy strategy only if each queue buffers at most $\lfloor B/2 \rfloor$ packets and, hence, the risk of packet loss is low. Thus we consider SGR to be a very practical algorithm. We analyze SGR by defining a new potential function that measures the number of packets that SGR has already lost or could lose if an adversary replenishes corresponding queues. In contrast to standard amortized analysis, we do not bound the potential change in each time step. Rather, we show that if the potential increased at T_1 time steps and $T_1 > C_1$ for some constant C_1 , then the potential must have decreased at T_2 steps with $T_2 > C_2$.

In the second part of the paper we also study the case when an online algorithm is granted more resources than an optimal offline algorithm and show that we can beat the competitiveness of 2. We consider resource augmentation with respect to *memory* and *speed*; i.e., we study settings in which an online algorithm has (a) larger buffers in each queue or (b) a higher transmission rate. For scenario (a) we prove that any work-conserving algorithm, and in particular any greedy algorithm, achieves a competitive ratio of $(c+2)/(c+1)$ if it has an additional buffer of $A = cB$ in each queue. We show that this bound is tight for greedy strategies. Hence, by doubling the buffer capacities we obtain a performance ratio of 1.5. For scenario (b) we show an upper bound of $1 + 1/k$ if in each time step an online algorithm can transmit k times as many packets as an adversary. Again, by doubling the transmission rate we obtain a competitiveness of 1.5. Finally, we give a linear time offline algorithm for computing an optimal service schedule maximizing the throughput.

This paper is organized as follows. In section 2 we develop our lower bounds. In section 3 we present the new SGR algorithm and investigate scenarios with resource augmentation. The optimal offline algorithm is given in section 4.

2. Lower bounds. We first analyze greedy algorithms and then develop lower bounds for arbitrary deterministic and randomized online strategies.

2.1. Greedy algorithms. Formally, we call an online algorithm GR *greedy* if GR always serves a longest queue. Greedy algorithms may differ in the way ties are broken in case several queues currently store a maximum number of packets. The tie-breaking rule may also be randomized.

THEOREM 1. *For any B , the competitive ratio of any randomized greedy algorithm GR is not smaller than $2 - 1/B$ if $m \gg B$.*

Proof. Fix a buffer size $B > 0$. We show that there exist infinitely many m and associated packet arrival sequences for m queues such that the throughput achieved by an adversary ADV is at least $2 - 1/B - \Theta(m^{-1/2^{B-2}})$ times that achieved by GR . This proves the theorem. We use arrival sequences, where whenever there are several queues of maximum lengths, all these queues are served once before the next packets arrive. Thus, the tie-breaking criteria need not be considered.

Let $\mu \geq 2$ be an integer and $b = 2^{B-2}$. Set $m = \mu^b$. We construct a recursive partitioning of the m queues. For any i with $1 \leq i \leq B-2$, let $m_i = m^{1/2^i}$. The m queues are divided into m_1 blocks, each consisting of m_1 subsequent queues. These blocks are labeled $1, \dots, m_1$ in ascending order. Block n_1 with $1 \leq n_1 \leq m_1$ is subdivided into m_2 blocks, each consisting of m_2 subsequent queues labeled $(n_1, 1), \dots, (n_1, m_2)$. This partitioning is repeated up to level $B-2$. In general, any block (n_1, \dots, n_i) at level i consisting of m_i queues is subdivided into m_{i+1} blocks each, containing m_{i+1} queues. These blocks are labeled $(n_1, \dots, n_i, 1), \dots, (n_1, \dots, n_i, m_{i+1})$. Note that a block (n_1, \dots, n_{B-2}) at level $B-2$ consists of exactly μ queues. We define a lexicographic

ordering on the $(B-2)$ -tuples (n_1, \dots, n_{B-2}) in the standard way. Given (n_1, \dots, n_{B-2}) and (n'_1, \dots, n'_{B-2}) we have $(n_1, \dots, n_{B-2}) < (n'_1, \dots, n'_{B-2})$ if $n_i < n'_i$, for some i and $n_j = n'_j$ for all $1 \leq j < i$. Furthermore, $(n_1, \dots, n_{B-2}) \leq (n'_1, \dots, n'_{B-2})$ if $(n_1, \dots, n_{B-2}) < (n'_1, \dots, n'_{B-2})$ or $n_i = n'_i$ for all $1 \leq i \leq B-2$.

The basic idea of the lower bound construction is to maintain a *staircase of packets* in GR 's queues, where we call a buffer configuration a staircase *centered at block* (n_1, \dots, n_{B-2}) if GR 's queues in any block (n'_1, \dots, n'_{B-2}) buffer i packets if $n_j = n'_j$, for $1 \leq j \leq i$, but $n_{i+1} \neq n'_{i+1}$. During our construction, the staircase center moves through the blocks in increasing lexicographic order. Note that at each such center move from (n_1, \dots, n_{B-2}) to its successor (n'_1, \dots, n'_{B-2}) , where $(n_1, \dots, n_i) = (n'_1, \dots, n'_i)$ is the greatest common label prefix of the two blocks, only the queues in blocks whose labels also have this prefix are affected.

When the center is located at (n_1, \dots, n_{B-2}) we force a packet loss of B at each of GR 's queues in that block. ADV will be able to accept all packets and essentially has full queues in all blocks (n'_1, \dots, n'_{B-2}) that are lexicographically smaller than (n_1, \dots, n_{B-2}) . When the construction ends, almost all of ADV 's queues are fully populated while GR 's queues are empty. Since a total of nearly mB packets are transmitted by ADV and GR during the construction, this gives the desired lower bound.

Formally, we process blocks (n_1, \dots, n_{B-2}) with $n_i \geq 2$ for all i in increasing lexicographic order. Blocks (n_1, \dots, n_{B-2}) with $n_i = 1$ for some i are special in that less than B packets will arrive there. When we start processing a block (n_1, \dots, n_{B-2}) with $n_i \geq 2$ for all i , certain invariants, given below, hold. We show how to process it such that the invariants are also true when we start processing the next block.

- (G1) Let (n'_1, \dots, n'_{B-2}) be a block with $(n'_1, \dots, n'_{B-2}) < (n_1, \dots, n_{B-2})$ and $n'_i \geq 2$ for all i . GR buffers exactly $j+1$ packets in each of the queues if j is the largest index with $n'_1 = n_1, \dots, n'_j = n_j$.
- (G2) Let (n'_1, \dots, n'_{B-2}) be a block with $(n'_1, \dots, n'_{B-2}) < (n_1, \dots, n_{B-2})$ such that $n'_i = 1$ and $n'_j \geq 2$ for all $j \leq i-1$. GR has $j+1$ packets in each of the queues if $n'_j = n_j$ for all $1 \leq j \leq i-1$.
- (G3) Let (n'_1, \dots, n'_{B-2}) be a block with $(n'_1, \dots, n'_{B-2}) \geq (n_1, \dots, n_{B-2})$. GR buffers exactly j packets in each of the queues if j is the largest index with $n'_1 = n_1, \dots, n'_j = n_j$.
- (A1) Let (n'_1, \dots, n'_{B-2}) be a block with $(n'_1, \dots, n'_{B-2}) < (n_1, \dots, n_{B-2})$ and $n'_i \geq 2$ for all i . ADV has B packets in each of the first $m_{B-2} - 1 = \mu - 1$ queues and $B-1$ packets in the last queue of this block.
- (A2) Let (n'_1, \dots, n'_{B-2}) be a block with $(n'_1, \dots, n'_{B-2}) < (n_1, \dots, n_{B-2})$ such that $n'_i = 1$ and $n'_j \geq 2$ for all $1 \leq j \leq i-1$. ADV has two packets in each of the queues if $n'_j = n_j$ for all $1 \leq j \leq i-1$ and one packet in those queues otherwise.
- (A3) Let (n'_1, \dots, n'_{B-2}) be a block with $(n'_1, \dots, n'_{B-2}) \geq (n_1, \dots, n_{B-2})$. ADV has 0 packets in each of those queues.

Initialization. We show how to establish the six invariants for the block $(2, \dots, 2)$, for which we illustrate in Figure 1 the initial configurations to be described next. Initially there arrive $2m_1$ packets in the queues of block (1) at level 1, with two packets in each queue, and m_1 packets in the queues of block (2) at level 1, with one packet in each queue. GR starts transmission from block 1 while ADV does so from block 2. After both GR and ADV have transmitted m_1 packets, we jump to level 2, where the arrival pattern is adopted in gauge: there arrive $2m_2$ packets in block $(2, 1)$, with two in each queue, and m_2 packets in block $(2, 2)$, with one in each

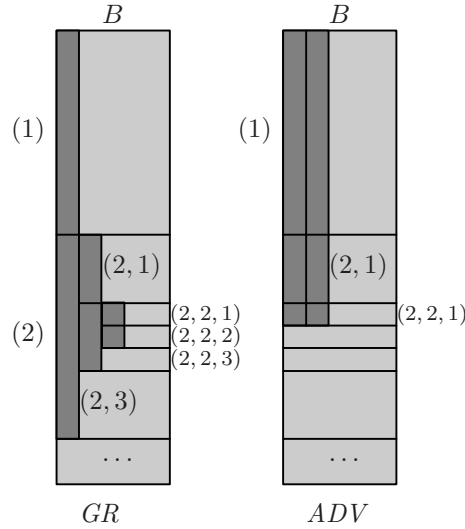


FIG. 1. Queue configurations when we start processing block $(2, 2, 2)$.

queue. We continue to copy and scale down this pattern until blocks $(2, \dots, 2, 1)$ and $(2, \dots, 2, 2)$ at level $B - 2$ are reached. At level i , GR always clears m_i packets in block $(2, \dots, 2, 1)$ while ADV does so in $(2, \dots, 2, 2)$.

Invariants (A1) and (G1) trivially hold because there is no block $N' < (2, \dots, 2)$ with $n'_i \geq 2$ for all $1 \leq i \leq B - 2$. If $N' < (2, \dots, 2)$ and i is the smallest index with $n'_i = 1$, then $n'_1 = \dots = n'_{i-1} = 2$. Queues in N' received $i + 1$ packets, $i - 1$ of which have been transmitted by ADV , while only one of them has been transmitted by GR . Hence, invariants (A2) and (G2) hold. If $N' \geq (2, \dots, 2)$ and j is the largest index with $n'_1 = n_1, \dots, n'_j = n_j$, then queues in N' received j packets, all of which have been transmitted by ADV , while none of them have been transmitted by GR , giving that invariants (A3) and (G3) hold.

Processing of a block. We next describe how to process block $N = (n_1, \dots, n_{B-2})$. Figure 2 shows the buffer configurations when the processing starts. Let $q_1, \dots, q_{m_{B-2}}$ be the m_{B-2} queues in that block. By (G3), GR has $B - 2$ packets in each of these queues. By (A3), ADV stores no packets there. We subsequently apply the following arrival pattern to all but the last of the q_j 's: There arrive B packets at queue q_j and one packet at queue q_{j+1} . First, GR accepts two packets in q_1 and one packet in q_2 . Then q_1 is completely populated while q_2 still has one free buffer cell left. Afterward, GR transmits a packet from q_1 . At the arrival of B packets at q_2 , GR must reject all but one of them and can accept the additional packet at q_3 as well. This behavior is repeated until the last queue in N is reached. In contrast, ADV always processes the single packet in q_{j+1} to be able to accept B packets in the next step. When B packets arrive at $q_{m_{B-2}}$, we omit the additional packet because we would cross the boundary to the next block of level $B - 2$. We assume that both ADV and GR then transmit one packet from $q_{m_{B-2}}$. Hence GR stores $B - 1$ packets in each queue of N , whereas ADV buffers B packets in all but the last queue and $B - 1$ packets in the last one. Next, we show how to establish the six invariants for the next block $\bar{N} = (\bar{n}_1, \dots, \bar{n}_{B-2})$ in lexicographic order satisfying $\bar{n}_i \geq 2$ for all i . We distinguish cases depending on whether or not $\bar{n}_{B-2} = n_{B-2} + 1$.

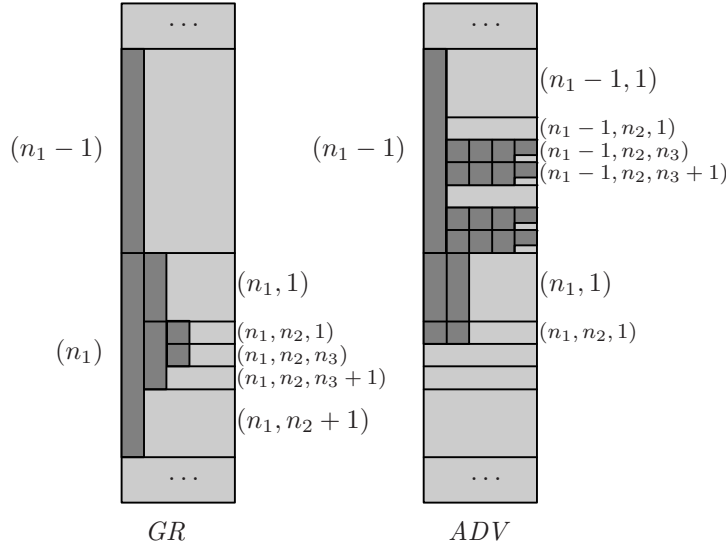


FIG. 2. Queue configurations when we start processing block (n_1, n_2, n_3) .

Case 1. If $\bar{n}_{B-2} = n_{B-2} + 1$, then N and \bar{N} belong to the same block of level $B - 3$. By (G3), GR buffers $B - 3$ packets in each queue of \bar{N} . Now there arrive m_{B-2} packets at \bar{N} , one at each of the queues. Thus, each queue in N buffers $B - 1$ packets, and each queue in \bar{N} buffers $B - 2$ packets. In the following m_{B-2} time steps, GR transmits one packet from each queue in N while ADV serves the queues in \bar{N} , which are then empty again in ADV 's configuration. Since only N and \bar{N} have been affected, the invariants (G1), (G2), and (G3) hold for \bar{N} as well. The same is true for (A1) because the statement holds for block N , as argued in the previous paragraph. (A2) was not affected during the processing of N because $\bar{n}_i \geq 2$ for all $1 \leq i \leq B - 2$. Since no new packets arrived at blocks that are lexicographically larger than \bar{N} , (A3) is also satisfied.

Case 2. If $\bar{n}_{B-2} \neq n_{B-2} + 1$, then N and \bar{N} do not belong to the same block at level $B - 3$. Let i be the largest index such that $n_i < m_i$; i.e., there is another block at level i . Hence $\bar{N} = (n_1, \dots, n_{i-1}, n_i + 1, 2, \dots, 2)$. In the following $\sum_{j=i+1}^{B-2} m_j$ time steps, no new packets arrive, and since (G1) and (G2) hold, GR transmits m_j packets from the queues of block $(n_1, \dots, n_i, m_{i+1}, \dots, m_j)$, for $j = B - 2, \dots, i + 1$. During each iteration, one packet is transmitted from each of these queues. In these time steps, for $j = B - 2, \dots, i + 1$, ADV transmits one packet from each of the queues in $(n_1, \dots, n_i, m_{i+1}, \dots, m_{j-1}, 1)$. By invariant (A2), these queues hold exactly two packets in ADV 's configuration and store precisely one packet after the transmission. In the next time step m_i packets arrive at the queues of $(n_1, \dots, n_i + 1)$, one packet at each of these queues. At that time in GR 's configuration, the queues in (n_1, \dots, n_i) buffer exactly $i + 1$ packets while all other queues buffer less. GR then transmits one packet from each of the queues in (n_1, \dots, n_i) while ADV serves the queues in $(n_1, \dots, n_i + 1)$ so that they are empty again. In the following $\sum_{j=i+1}^{B-2} m_j$ time steps, we restore in GR 's configuration the full staircase on top of the i packets in the queues of $(n_1, \dots, n_i + 1)$. More precisely, there arrive $2m_{i+1}$ packets at the queues of $(n_1, \dots, n_i + 1, 1)$, two at each of these queues, and m_{i+1} packets at the

queues of $(n_1, \dots, n_i + 1, 2)$, one packet at each of these queues. GR transmits one packet from each queue in $(n_1, \dots, n_i + 1, 1)$ while ADV clears block $(n_1, \dots, n_i + 1, 2)$. Then there arrive $2m_{i+2}$ packets in $(n_1, \dots, n_i + 1, 2, 1)$ and m_{i+2} packets in $(n_1, \dots, n_i + 1, 2, 2)$. Again GR serves the queues in the first of these blocks while ADV clears the second. This process continues up to blocks $(n_1, \dots, n_i + 1, 2, \dots, 2, 1)$ and $(n_1, \dots, n_i + 1, 2, \dots, 2)$ at level $B - 2$.

LEMMA 1. *Invariants (G1–3) and (A1–3) hold when we start processing $\bar{N} = (n_1, \dots, n_i + 1, 2, \dots, 2)$.*

Proof. Consider block $\bar{N} = (n_1, \dots, n_i + 1, 2, \dots, 2) =: (\bar{n}_1, \dots, \bar{n}_{B-2})$. Let $N' = (n'_1, \dots, n'_{B-2})$ be an arbitrary block. We first study (G1). Let $N' < N$, $n'_j \geq 2$ for all j , and let k be the largest index such that $n'_1 = n_1, \dots, n'_k = n_k$. If $k < i$, then there is nothing to show because the queues in N' have not been touched by GR since the processing of N started. If $k \geq i$, we have $N' = (n_1, \dots, n_i, m_{i+1}, \dots, m_k, n'_{k+1}, \dots, n'_{B-2})$. Thus N' is affected at the iteration steps for $j = k, \dots, i$, and hence $k - i + 1$ times, where iteration i corresponds to the subsequent transmission of one packet from each queue in (n_1, \dots, n_i) . Since N' buffered $k + 1$ packets before the processing of N started, there are $k + 1 - (k - i + 1) = i$ packets after the iteration steps. On the other hand, $i - 1 = \max\{j : n'_1 = \bar{n}_1, \dots, n'_j = \bar{n}_j\}$. For $N' = N$ the statement of (G1) also holds because exactly i packets are buffered at these queues. If $N' < N$, statement (G2) holds because of the same arguments, starting with k packets before the processing and eventually getting i packets.

If $N < N' < \bar{N}$, then let j be the largest index with $n'_1 = \bar{n}_1, \dots, n'_j = \bar{n}_j$. We have $i \leq j < B - 2$ and $n'_{j+1} = 1$. Since $n'_{i+1} = \dots = n'_j = 2$, GR buffers exactly $j + 1$ packets in the queues of N' . Hence (G2) holds here as well. Moreover, since GR has $B - 2$ packets in the queues of \bar{N} , (G3) holds for $N' = \bar{N}$. If $N' > \bar{N}$, then we distinguish two cases. If $n'_l > \bar{n}_l$ for some $1 \leq l \leq i$, there is nothing to show because GR 's configuration in these queues has not changed and the largest index j with $n_1 = n'_1, \dots, n_j = n'_j$ is equal to the largest index j with $\bar{n}_1 = n'_1, \dots, \bar{n}_j = n'_j$. If $n'_1 = \bar{n}_1, \dots, n'_i = \bar{n}_i = n_i + 1$, then let j be the largest index with $n'_1 = \bar{n}_1, \dots, n'_j = \bar{n}_j$. We have $n'_{i+1} = 2, \dots, n'_j = 2$ and $n'_{j+1} > 2$. Hence the queues in N' store exactly j packets and (G3) holds.

Invariant (A1) obviously holds because it held when we started processing N and the desired property was established for block N . There exist no blocks N' with $N < N' < \bar{N}$ and $n'_i \geq 2$ for all i . Invariant (A2) is satisfied for blocks N' with $N' \leq N$ because, during the processing of N , ADV served the queues in $(n_1, \dots, n_i, m_{i+1}, \dots, m_{j-1}, 1)$ for $j = B - 1, \dots, i + 1$ exactly once, thus reducing the number of packets stored there from 2 to 1. If $N' > N$, then the queues store exactly two packets, as desired. Finally, (A3) holds because ADV has transmitted all packets that have arrived at blocks $N' \geq \bar{N}$. \square

After processing the last block (m_1, \dots, m_{B-2}) , no further packets arrive, but the iteration steps are executed as if there were another block. Then, we have the following configurations: From invariants (G1), (G2), and (G3), we derive that GR buffers one packet in each queue. Due to (A1), (A2), and (A3), ADV buffers $B - 1$ or B packets in the queues in blocks (n_1, \dots, n_{B-2}) with $n_i \geq 2$ for all i while the others buffer exactly one packet like GR does.

Let T_G be the throughput achieved by GR and T_A be the throughput achieved by ADV . For any block (n_1, \dots, n_{B-2}) with $n_i \geq 2$ for all i , GR transmits B packets from each of the associated queues. For any block $(n_1, \dots, n_{i-1}, 1)$ with $n_j \geq 2$, for $j = 1, \dots, i - 1$, GR transmits $i + 1$ packets from each queue. There are $\prod_{j=1}^{i-1} (m_j - 1)$

such blocks, each containing m_i queues. Thus $T_G = (m - \check{m})B + \delta_1$, where

$$\check{m} = \sum_{i=1}^{B-2} \prod_{j=1}^{i-1} (m_j - 1)m_i \quad \delta_1 = \sum_{i=1}^{B-2} \prod_{j=1}^{i-1} (m_j - 1)m_i(i + 1).$$

The throughput of *ADV* is equal to that of *GR* plus the number of packets that are in *ADV*'s final configuration when the processing of blocks ends minus the number of packets that are in *GR*'s final configuration when the processing of blocks ends. In this final configuration, queues in blocks (n_1, \dots, n_{B-2}) with $n_i \geq 2$ for all i store B packets, except for the last of these queues, which buffers only $B - 1$ packets. All other queues are empty. Hence, $T_A = T_G + (m - \check{m})B - \delta_2 - (m - \check{m})$, where $\delta_2 = (m - \check{m})/\mu$. Hence $T_A \geq T_G + (m - \check{m})(B - 1) - m\mu^{-1}$. Moreover, $\check{m} \in \Theta(\prod_{j=1}^{B-2} m_j) = \Theta(\prod_{j=1}^{B-2} m^{\frac{1}{2^j}}) = \Theta(m^{1 - \frac{1}{2^{B-2}}})$, $m\mu^{-1} \in \Theta(m^{1 - \frac{1}{2^{B-2}}})$, and $\delta_1 \in \Theta(B \prod_{j=1}^{B-2} m_j) = \Theta(Bm^{1 - \frac{1}{2^{B-2}}})$. This implies

$$\begin{aligned} \frac{T_A}{T_G} &\geq \frac{(m - \check{m})B + \delta_1 + (m - \check{m})(B - 1) - m\mu^{-1}}{(m - \check{m})B + \delta_1} = 2 - \frac{(m - \check{m}) + \delta_1 + m\mu^{-1}}{(m - \check{m})B + \delta_1} \\ &\geq 2 - \frac{1}{B} - \frac{\delta_1 + m\mu^{-1}}{(m - \check{m})B + \delta_1} = 2 - \frac{1}{B} - \Theta\left(\frac{Bm^{1 - \frac{1}{2^{B-2}}}}{mB}\right) \\ &= 2 - \frac{1}{B} - \Theta\left(m^{-\frac{1}{2^{B-2}}}\right). \quad \square \end{aligned}$$

2.2. Arbitrary algorithms. We first study deterministic online algorithms and then address randomized strategies.

THEOREM 2. *The competitive ratio of any deterministic online algorithm ALG is at least $e/(e - 1)$ if $m \gg B$.*

Proof. Let B be a positive integer representing the buffer size. For any positive integer N , let $m = (B + 1)^N$ be the number of queues. Let the B buffer columns be indexed $1, \dots, B$, where column B is the one at the head of the queues and column 1 is the one at the tails. At the beginning there arrive B packets at each of the m queues such that all buffers are fully populated. We present a scheme S for constructing request sequences σ . Our scheme has the property that the throughput achieved by an adversary is at least $e/(e - 1)$ times the throughput obtained by any greedy algorithm and that the throughput of any greedy strategy is at least as large as the throughput of any deterministic online algorithm *ALG*. This establishes the theorem. It will be sufficient to consider the standard greedy algorithm, denoted by *GR*, which serves the smallest indexed queue in case there is more than one queue of maximum length.

A request sequence σ consists of superphases P_1, \dots, P_B . Superphase P_i consists of phases $(i, N), \dots, (i, 1)$. In a phase (i, s) essentially $\lfloor (B + 1)^s (\frac{B+1}{B})^{i-2} \rfloor$ packets arrive at the $q_s = (B + 1)^{s-1}$ most populated queues in the online configuration. We first present an informal description of the superphases and phases, explaining how *GR* would serve them. Then we give a formal definition with respect to any online strategy. When superphase P_i is finished, the last i columns in *GR*'s buffers are empty while the other columns are fully populated. In particular, when σ ends, all buffers are empty. Superphase P_i is meant to empty column i . After phase (i, s) the first $m - q_s$ queues contain $B - i$ packets while the remaining queues buffer $B - i + 1$ packets. This load difference of one packet in the last q_s queues compared to the others is sufficient for *GR* to serve each of these q_s queues exactly once during the next q_s time steps.

Since GR is a deterministic algorithm, the adversary knows, for each $1 \leq k \leq q_s$, which of the q_s queues are served by GR during the first k time steps of the next phase if no new packets arrive, and uses this knowledge to serve other queues and make new packets arrive there to cause as large as possible a packet loss for GR .

We first describe superphase P_1 and start with phase $(1, N)$. Initially, all $q_{N+1} = m$ queues are fully populated. During the first $q_N B$ time steps no packets arrive and GR transmits a packet from each of the first $q_N B$ queues. Then B packets arrive at each of the remaining q_N queues, all of which must be dropped by GR because the last q_N queues already buffer B packets each. An adversary, on the other hand, could transmit all packets from the last q_N queues during the first $q_N B$ time steps so that no packet loss occurs. At the end of phase $(1, N)$ the last q_N queues are fully populated in GR 's buffer configuration. The arrival pattern now repeats for the other phases in P_1 . At the beginning of $(1, s)$ the last q_{s+1} queues in GR 's configuration store B packets each. During the first $q_s B$ time steps no packets arrive and GR transmits one packet from each of the first $q_s B$ of these q_{s+1} queues. Then B packets are sent to each of the last q_s queues, all of which are lost by GR . Again, an adversary can accept all packets by transmitting from the last q_s queues in the previous time steps. At the end of $(1, 1)$ the last queue in GR 's configuration contains B packets while all other queues buffer $B - 1$ packets. Now there is one time step without packet arrivals such that GR can transmit one packet from the last queue and has exactly $B - 1$ packets in each of its buffers.

Superphase P_2 is similar. In $(2, N)$ there are no packet arrivals during the first $q_N B$ time units. Then there arrive B packets at each of the last q_N queues. This time GR loses $B - 1$ packets at each of the last queues, which are then fully populated again. To these last q_N queues we recursively apply the packet arrival pattern used to empty column 1 in P_1 . In phase $(2, s)$ there are no packet arrivals during the first $q_s B$ time units. Then B packets are sent to the last q_s queues, causing a packet loss of $(B - 1)$ at each of these buffers. To empty the last column of these queues, we recursively apply the pattern of P_1 . In general, in any phase (i, s) of P_i there are $q_s B$ time units without packet arrivals, followed by the arrival of B packets at each of the last q_s queues. GR loses $B - i + 1$ packets at each of these buffers, which are then fully populated again. To empty the last $i - 1$ columns of these buffers we recursively apply the pattern used in P_{i-1} .

Formally, for any deterministic online algorithm ALG , $\sigma = P_1, \dots, P_B$, where $P_i = (i, N), \dots, (i, 1)$. We call the BN phases in P_1, \dots, P_B *main phases*. A main phase triggers *auxiliary phases*. Auxiliary phases are identified by their complete triggering path. If φ_1 is a main phase and φ_i triggers φ_{i+1} , then the auxiliary phase φ_n is denoted by $[\varphi_1 \varphi_2, \dots, \varphi_n]$. Let Φ be the set of phases. To identify sets of queues on which phases work, we define a predecessor mapping $\pi : \Phi \rightarrow \Phi$. For a main phase $\varphi = (i, s)$, let i be the level of φ . If φ is a main phase, then $\pi(\varphi)$ is the immediately preceding phase in σ of the same level if such exists; otherwise we define $\pi(\varphi) = (0, 0)$. If φ is an auxiliary phase, then $\pi(\varphi)$ is the triggering phase:

- $\pi((i, N)) = (0, 0)$ and $\pi((i, s)) = (i, s + 1)$ if $s < N$,
- $\pi([(i_1, s_1), \dots, (i_n, s_n)]) = [(i_1, s_1), \dots, (i_{n-1}, s_{n-1})]$ if $s_n = s_{n-1} - 1$,
- $\pi([(i_1, s_1), \dots, (i_n, s_n)]) = [(i_s, s_1), \dots, (i_n, s_n + 1)]$ if $s_n < s_{n-1} - 1$.

Furthermore, we need a recursive mapping $Q : \Phi \rightarrow \mathcal{P}(\{1, \dots, m\})$ that associates phases with sets of queues. Let $Q(0, 0)$ be the set of all m queues.

Each phase φ with suffix (i, s) consists of the following steps.

1. $q_s B$ time steps without packet arrival, i.e., only packet transmission.

2. Arrival of B packets at each of the q_s most populated queues in $Q(\pi(\varphi))$; these queues form $Q(\varphi)$.
3. If $i > 1$ and $s > 1$, triggering of the phases $[\varphi, (1, s - 1)], \dots, [\varphi, (1, 1)], \dots, [\varphi, (i - 1, s - 1)], \dots, [\varphi, (i - 1, 1)]$ on $Q(\varphi)$.
4. If $s = 1$, then i time steps without packet arrival.

LEMMA 2. *If a sequence of phases $(1, s), \dots, (1, 1), \dots, (i, s), \dots, (i, 1)$ is served by GR on a set $Q_{G,s}$ consisting of $q_{s+1} = (B + 1)^s$ consecutive queues, then after phase (j, r) the buffer configuration in $Q_{G,s}$ is as follows. If $r > 1$, then the last q_r queues buffer $B - j + 1$ packets while the other queues buffer $B - j$ packets. If $r = 1$, then all queues buffer $B - j$ packets.*

Proof. We first prove the statement of the lemma under the condition that queues not contained in $Q_{G,s}$ buffer less than $B - i$ packets. Then we show that the condition is always satisfied.

The proof is by induction on i . First consider $i = 1$. At the beginning all buffers in $Q_{G,s}$ are full because the sequence is either a sequence of main phases started with fully populated buffers or a sequence of triggered phases, which are also initiated on full buffers only. In $(1, s)$ during the first $q_s B$ time steps, GR transmits a packet from the first $q_s B$ queues in $Q_{G,s}$ because queues outside $Q_{G,s}$ contain less than B packets. In the next time step the packets arriving at the last q_s queues in $Q_{G,s}$ are rejected because the queues already buffer B packets. No further phases are triggered in $(1, s)$. Thus the last q_s queues buffer B packets while the other queues store $B - 1$ packets. If $s = 1$, then $q_s = 1$ and in the last time step in $(1, 1)$ a packet is transmitted from the last queue so that all queues buffer exactly $B - 1$ packets. The desired statement on the buffer population holds after phase $(1, s)$. Suppose inductively that it holds after $(1, r + 1)$. Then the last q_{r+1} queues in $Q_{G,s}$ store B packets while the remaining queues each have $B - 1$ packets. During the next $q_r B$ time steps in $(1, r)$ one packet is transmitted from the first $q_r B$ queues among the last q_{r+1} buffers in $Q_{G,s}$. Thus the last q_r queues in $Q_{G,s}$ still have B packets while the remaining queues buffer $B - 1$ packets. Again, if $r = 1$, one packet is transmitted from the last queue in the final time steps in $(1, r)$ so that all queues have exactly $B - 1$ packets. Thus the stated buffer population is maintained after phase $(1, r)$ and the desired statement holds for $i = 1$.

Suppose that the statement holds for every integer smaller than i ; we prove that it is also satisfied for i . As above, we can show that after phase $(1, s)$ the buffer population is as desired. Assume that the statement on the buffer population holds up to but excluding phase (j, r) . At the beginning of (j, r) the last q_{r+1} buffers in $Q_{G,s}$ store $B - j + 1$ packets while the remaining queues contain $B - j$ packets. During the next time steps, GR transmits one packet from each of the first $q_r B$ buffers among the q_{r+1} last ones in $Q_{G,s}$ because buffers outside $Q_{G,s}$ store less than $B - i < B - j + 1$ packets. Thus the last q_r queues in $Q_{G,s}$ store $B - j + 1$ packets while the remaining queues each contain $B - j$ packets. Then B packets arrive at the last q_r queues so that they are fully populated again. If $r = 1$, then no phases are triggered and, since $q_1 = 1$, only the last queue is fully populated. During the next j time steps j packets are transmitted from this last queue and all queues in $Q_{G,s}$ then buffer $B - j$ packets. The queues are populated as desired. If $r > 1$, then phases $(1, r - 1), \dots, (1, 1), \dots, (j - 1, r - 1), \dots, (j - 1, 1)$ are triggered on the last q_r fully populated queues. The other queues buffer less than $B - (j - 1)$ packets. By an induction hypothesis, when the triggered phases end, the last q_r queues again buffer $B - j + 1$ packets. Thus the buffers are populated as desired and the desired

statement holds for every integer i .

It remains to show that the condition mentioned at the beginning of the proof holds; i.e., if a sequence of phases $(1, s), \dots, (1, 1), \dots, (i, s), \dots, (i, 1)$ is served on a set $Q_{G,s}$ of queues, then all queues not contained in $Q_{G,s}$ store less than $B - i$ packets. The condition holds when the initial sequence σ of main phases is started. Suppose that the condition holds for all sequences triggered after the beginning of σ but before the start of $(1, s), \dots, (1, 1), \dots, (i, s), \dots, (i, 1)$. Assume that the latter sequence is triggered by phase $[\varphi_1, \dots, \varphi_n]$ such that $\varphi_j = (i_j, s_j)$ and phase φ_j is executed on queues in Q_{φ_j} . We have $Q_{\varphi_{j+1}} \subset Q_{\varphi_j}$ for $j = 1, \dots, n - 1$, and queues outside Q_{φ_j} contain less than $B - i_j$ packets. Thus all queues outside Q_{φ_n} buffer less than $B - i_n$ packets. The sequence is triggered on the last q_{s_n} buffers in Q_{φ_n} . Since the first $q_{s_n} B$ buffers in Q_{φ_n} store $B - i_n$ packets and $i_1 > i_2 > \dots > i_n > i$, the sequence is triggered on a set of buffers storing less than $B - i$ packets. The proof is complete. \square

LEMMA 3. *The packet loss of any deterministic online algorithm ALG is not smaller than the one of GR.*

Proof. Before presenting the details of the proof, we outline the basic idea. We construct a request sequence σ that is similar in structure to the sequence used for GR in the previous lemma. While GR gives us a hierarchical structure for the distinct phases, ALG might serve the queues in such a way that the set of the most populated queues varies over time. Therefore, we generalize the request sequence such that, upon each packet arrival, the new packets arrive at the currently most populated queues. In the sequence to be constructed, there are packet arrivals at exactly the same time steps as for GR . Moreover, the arrival pattern is not changed either; i.e., for each time step there is a match between GR 's queues and ALG 's queues such that the same number of packets arrives at each pair of matched queues. We shall show that the number of packets transmitted by ALG is at most the number of packets transmitted by GR .

To establish the claim, we use Lemma 2. We consider sequences $\sigma(i, s) = (1, s), \dots, (1, 1), \dots, (i, s), \dots, (i, 1)$ processed by ALG and GR on sets Q_A and Q_G of q_s buffers each. For any given time since the beginning of $\sigma(i, s)$, let N_G be the total number of packets in GR 's queues of Q_G and let L_G be the total packet loss of GR accumulated since the start of $\sigma(i, s)$. For algorithm ALG , N_A and L_A are defined similarly. Furthermore, let S_A be the number of time steps in $\sigma(i, s)$ where neither do packets arrive nor does ALG transmit a packet from queues in Q_A . We are interested in the following invariants:

$$(I1) \quad N_A - S_A + L_A = N_G + L_G, \quad (I2) \quad N_A - S_A \leq N_G.$$

We prove the following statement. If a sequence $\sigma(i, s)$ is processed by ALG and GR on sets Q_A and Q_G , respectively, then after each phase, invariants (I1) and (I2) hold. The lemma then follows by considering a sequence $\sigma = \sigma(B, N)$. At the end of the sequence, by (I1), $L_A = N_G + L_G - (N_A - S_A)$. By (I2) we conclude $L_A \geq L_G$.

We prove the desired statement by induction on i . First consider $i = 1$. At the beginning of $\sigma(1, s)$ all queues in Q_A and Q_G are fully populated because sequence σ as well as triggered phases are initiated on full buffers only. Therefore $N_A = N_G$. Since initially $L_A = L_G = 0$ and $S_A = 0$, both invariants hold at the beginning of $\sigma(1, s)$. While serving $\sigma(1, s)$, in each time step GR transmits a packet from queues in Q_G . Moreover, GR always serves a fully populated queue. Therefore, at any time, GR always has the smallest number of fully populated queues. By Lemma 2, at the

beginning of phase $(1, r)$ the last q_{r+1} queues in Q_G buffer B packets while all other queues store $B-1$ packets. After the next $q_r B$ time steps, the last q_r queues still buffer B packets such that all packets arriving in step 2 of $\sigma(1, r)$ are lost. Since ALG has at least q_r fully populated queues, ALG also loses all incoming packets. Therefore, at any time the packet loss of ALG and GR is the same. Since GR transmits a maximum number of packets and no incoming packets can be accepted by ALG and GR , $N_A \geq N_G$. If $N_A - N_G = \delta > 0$, then there must have been δ time units, where ALG did not transmit a packet. Therefore (I1) and (I2) hold throughout the execution of $\sigma(1, s)$.

Assume that the statement to be proven holds for integers smaller than i . We consider a sequence $\sigma(i, s)$. Again (I1) and (I2) hold initially, when $\sigma(i, s)$ is started. We show that if the invariants (I1) and (I2) hold before any phase (j, r) , then they also hold after the phase. At the beginning of (j, r) , GR buffers $B - j + 1$ packets in the last q_{r+1} queues of Q_G and $B - j$ packets in the remaining queues. During the first $q_r B$ time steps, GR always transmits one packet. Consider any time step. If ALG also transmits a packet from queues in Q_A , then both N_A and N_G decrease by 1. If ALG does not serve a queue in Q_A , then only N_G decreases by 1 but S_A increases by 1. The invariants are maintained. Immediately before step 2 of phase (j, r) let Q_A^1 be the set of the q_r most populated buffers in Q_A to which packets are sent in step 2. Set $Q_A^2 = Q_A \setminus Q_A^1$ and let N_A^i be the number of packets in $Q_A^i, i = 1, 2$. Similarly, let Q_G^1 be the set of the q_r most populated queues in $Q_G, Q_G^2 = Q_G \setminus Q_G^1$ and N_G^i be the packet number in $Q_G^i, i = 1, 2$. Each queue in Q_G^2 has exactly $B - j$ packets while each queue in Q_G^1 buffers exactly $B - j + 1$ packets. We have $N_A^2 - S_A \leq N_G^2$. Otherwise, if $N_A^2 - S_A > N_G^2$, then the most populated queues in Q_A^2 would store at least $B - j + 1$ packets. Hence $N_A^1 \geq N_G^1$ and (I1) would be violated.

Now consider the effect of step 2 of phase (j, r) in which the queues of Q_A^1 and Q_G^1 are refilled. If $L_A - L_G$ changes by δ , then $N_G - N_A$ changes by δ so that (I1) is maintained. Since $N_A^1 = N_G^1$ after the step and $N_A^2 - S_A \leq N_G^2$, invariant (I2) also holds. If $r = 1$, then no phases are triggered and there are j more time steps with packet transmission in which the invariants are maintained. If $r > 1$, then phases $(1, r - 1), \dots, (1, 1), \dots, (j - 1, r - 1), \dots, (j - 1, 1)$ are triggered in step 3 of (j, r) . The triggered phases are executed on Q_A^1 and Q_G^1 . Since $N_A^1 = N_G^1$ initially, the desired invariants hold at the beginning of the triggered sequence. By an induction hypothesis we have $N_A^1 - S_A^1 + L_A^1 = N_G^1 + L_G^1$ and $N_A^1 - S_A^1 \leq N_G^1$ when the sequence is finished. Here L_A^1 and L_G^1 denote the loss incurred during the sequence and S_A^1 is the number of time steps, ALG does not transmit packets from Q_A^1 . Let $S_A^{1,1}$ be the number of time steps in the triggered sequence where ALG works on neither Q_A^1 nor Q_A^2 , and let $S_A^{1,2}$ be the number of time steps where ALG does not work on Q_A^1 but does transmit packets from Q_A^2 . Then $S_A^1 = S_A^{1,1} + S_A^{1,2}$. Before step 3 of the phase, (I1) held and $N_A^1 = N_G^1$. Therefore $N_A^2 - S_A + L_A = N_G^2 + L_G$ and, as argued above, $N_A^2 - S_A \leq N_G^2$. Using the invariants for the triggered sequence, we obtain

$$(N_A^1 + N_A^2 - S_A^{1,2}) - (S_A + S_A^{1,1}) + (L_A + L_A^1) = (N_G^1 + N_G^2) + (L_G + L_G^1)$$

and

$$(N_A^1 + N_A^2 - S_A^{1,2}) - (S_A + S_A^{1,1}) \leq (N_G^1 + N_G^2).$$

The invariants thus hold at the end of phase (j, r) because $N_A^1 + N_A^2 - S_A^{1,2}$ is the total number of packets in queues from $Q_A; S_A + S_A^{1,1}$ is the total number of time

steps ALG does not transmit packets from Q_A ; and $N_G^1 + N_G^2$ is the number of packets in Q_G . Finally, $L_A + L_A^1$ is the total loss accumulated by ALG and $L_G + L_G^1$ is the corresponding value of GR . The inductive proof is complete. \square

LEMMA 4. *The throughput of the adversary ADV is at least $e/(e - 1)$ times that of GR .*

Proof. We analyze the competitive ratio of the GR algorithm. To this end we analyze the throughput T_{ADV} achieved by an adversary ADV and the throughput T_G of GR . At the beginning of σ , all queues are fully populated. When σ ends, by Lemma 2, all of GR 's buffers are empty. We describe how ADV serves σ . In phase (i, s) during the first $q_s B$ time steps ADV transmits all packets from the q_s queues at which packets arrive in step 2 of the phase. If $s = 1$, then in step 4 of the phase ADV is idle and does not transmit packets. Thus in each phase ADV can always accept all incoming packets and at the end of the phase all buffers are full. Let S_{ADV} be the total number of time steps where ADV is idle in σ . Since GR transmits a packet in each time step and ADV 's queues contain mB packets at the end of σ , we have $T_G = T_{ADV} - mB + S_{ADV}$.

We estimate S_{ADV} . The adversary is idle in a phase $\varphi = [\varphi_1, \dots, \varphi_n]$ with $\varphi_j = (i_j, s_j)$ if $s_n = 1$. Since $i_1 > \dots > i_n$ and $i_j > 1$ for $j < n$, there are $\sum_{n=1}^B \binom{B}{n} (N - 1)^{n-1} \leq N^B$ such phases, in each of which ADV is idle for at most B time units. Thus $S_{ADV} \leq BN^B$ and the competitive ratio of GR is at least

$$c \geq T_{ADV}/T_G \geq 1 / \left(1 - \frac{mB}{T_{ADV}} + \frac{N^B B}{T_{ADV}} \right) \geq 1 / \left(1 - \frac{mB}{T_{ADV}} + \frac{N^B}{m} \right)$$

because $T_{ADV} \geq mB$. To lower bound the last expression we have to upper bound T_{ADV} . The throughput of ADV is equal to the mB packets initially buffered in the queues plus the packets accepted by ADV during the service of σ . As argued above, ADV can always accept all packets. Let $a_{i,s}$ be the number of packets accepted by ADV in a phase (i, s) . We show by induction on i that $a_{i,s} \leq (B + 1)^s \left(\frac{B+1}{B}\right)^{i-2}$. The inequality holds for $i = 1$ because $a_{1,s} = q_s B = (B + 1)^{s-1} B$. Suppose that the inequality holds for i . We have $a_{i+1,s} = q_s B + \sum_{j=1}^i \sum_{r=1}^{s-1} a_{j,r} = a_{i,s} + a_{i,s-1} + \dots + a_{i,1} \leq \left(\frac{B+1}{B}\right)^{i-2} \sum_{r=1}^s (B + 1)^r \leq \left(\frac{B+1}{B}\right)^{i-1} (B + 1)^s$. The total number of accepted packets in σ is at most

$$\begin{aligned} \sum_{i=1}^B \sum_{s=1}^N a_{i,s} &\leq \sum_{i=1}^B \left(\frac{B+1}{B}\right)^{i-2} \frac{(B+1)^{N+1} - 1}{B} \leq (B+1)^N \sum_{i=1}^B \left(\frac{B+1}{B}\right)^{i-1} \\ &= (B+1)^N B \left(\left(1 + \frac{1}{B}\right)^B - 1 \right) = mB \left(\left(1 + \frac{1}{B}\right)^B - 1 \right). \end{aligned}$$

Since the initial buffer configuration stores mB packets, we have $T_{ADV} \leq mB(1 + \frac{1}{B})^B$ and we conclude $c \geq 1/(1 - (1 + 1/B)^{-B} + N^B/m)$ and for large B and N this expression can be arbitrarily close to $e/(e - 1)$. \square

Lemmas 3 and 4 establish the theorem. \square

Azar and Richter [4] proved a lower bound of 2 for all deterministic online algorithms if $B = 1$. In the previous theorem we presented a construction of a lower bound that converges to $e/(e - 1)$ if $B \rightarrow \infty$. This raises the question of whether the competitive ratio can be bounded away from both $e/(e - 1)$ and 2 if $B > 1$, but small. For $B = 2$ we can show such a stronger bound, which we prove to be optimal in section 3 by presenting a matching upper bound.

THEOREM 3. *For $B = 2$, no deterministic online algorithm can achieve a competitive ratio smaller than $13/7$.*

Proof. We enhance the request sequence σ presented in the proof of Theorem 2. Let there be m additional queues for both ADV and ALG . At the beginning, there arrives one packet in each of the $2m$ queues. During the first m time steps, ADV serves those queues not served by ALG during this period. Then, σ is applied to those queues already served by ADV . Let T_{ADV} and T_{ALG} denote the throughputs of ADV and ALG , respectively, for σ , again, and let T'_{ADV} and T'_{ALG} denote the respective throughputs for the composed request sequence. Since $T'_{ADV} = T_{ADV} + 2m$ and $T'_{ALG} = T_{ALG} + m$, the throughput difference for any online algorithm is the same as for GR . Hence, we can apply Lemma 3 again and adopt the proof of Lemma 4. Let T_G and T'_G denote the respective throughputs for GR . Since $T'_G = T'_{ADV} - (m + 1)B + S_{ADV}$ and $mB \leq T'_{ADV} \leq m(B(1 + \frac{1}{B})^B + 2)$, we get for the competitive ratio that

$$\begin{aligned} c \geq T'_{ADV}/T'_G &\geq 1 / \left(1 - \frac{m(B+1)}{T_{ADV}} + \frac{N^B}{m} \right) \\ &\geq 1 / \left(1 - (B+1) \left(B \left(1 + \frac{1}{B} \right)^B + 2 \right)^{-1} + \frac{N^B}{m} \right), \end{aligned}$$

taking values arbitrarily close to $1/(1 - (B+1)(B(1 + \frac{1}{B})^B + 2)^{-1})$ for large N . Putting $B = 2$ yields $c \geq 1/(1 - 3(2(1 + \frac{1}{2})^2 + 2)^{-1}) = 1/(1 - 3(\frac{13}{2})^{-1}) = 1/(1 - \frac{6}{13}) = \frac{13}{7}$. \square

We next consider randomized algorithms. The advantage of randomization consists of the online algorithm's being more powerful against the adversary than in the deterministic case because the former can use its probability distribution to fool the latter. For packet buffering, Azar and Richter [4] showed that if $B = 1$, no randomized online algorithm can achieve a competitive ratio smaller than 1.4659. We prove the same lower bound for any buffer size B . This reveals a significant difference from deterministic algorithms, where the lower bound of 2 shown for $B = 1$ does not hold for any other B .

THEOREM 4. *If ALG is a randomized online algorithm, its competitive ratio cannot be smaller than 1.4659 for any buffer size B .*

Proof. Let each queue be populated by B packets at the beginning. We call B subsequent time steps a round. At the end of each round, B packets arrive in the queue currently having the maximum expected load. Since the adversary ADV knows where the next packets will arrive, it serves this queue during the preceding B time steps. Hence, ADV can accept all packets, whereas ALG has an expected loss of l_i packets in round i . First, we show that

$$(1) \quad \sum_{i=1}^n l_i \geq \sum_{i=1}^n \left(\frac{m-1}{m} \right)^i B.$$

Then, we will derive three lemmas from which we eventually conclude our claim.

We show (1) by induction on i . Let $i = 1$. After B time steps, ALG has a population of $mB - B = (m - 1)B$ packets. Hence, the average queue length is $\frac{m-1}{m}B$. So, the maximum expected queue length is at least $\frac{m-1}{m}B$, yielding an expected loss $l_1 \geq \frac{m-1}{m}B$. Now assume $\sum_{i=1}^j l_i \geq \sum_{i=1}^j \left(\frac{m-1}{m} \right)^i B$. Until the end of round $j + 1$, ALG has transmitted $(j + 1)B$ packets while its expected number of accepted packets

is $jB - \sum_{i=1}^j l_i$. Hence, its expected buffer population is

$$mB - (j+1)B + jB - \sum_{i=1}^j l_i = (m-1)B - \sum_{i=1}^j l_i.$$

So, the expected average queue length is

$$\frac{1}{m} \left((m-1)B - \sum_{i=1}^j l_i \right) = \frac{m-1}{m}B - \frac{1}{m} \sum_{i=1}^j l_i.$$

We derive that the maximum expected queue length after round $j+1$ is at least $\frac{m-1}{m}B - \frac{1}{m} \sum_{i=1}^j l_i$, yielding an expected loss $l_{j+1} \geq \frac{m-1}{m}B - \frac{1}{m} \sum_{i=1}^j l_i$. Hence

$$\begin{aligned} \sum_{i=1}^{j+1} l_i &\geq \sum_{i=1}^j l_i + \frac{m-1}{m}B - \frac{1}{m} \sum_{i=1}^j l_i = \frac{m-1}{m} \left(B + \sum_{i=1}^j l_i \right) \\ &\geq \frac{m-1}{m} \left(B + \sum_{i=1}^j \left(\frac{m-1}{m} \right)^i B \right) = \sum_{i=1}^{j+1} \left(\frac{m-1}{m} \right)^i B. \end{aligned}$$

If there are n rounds, then the throughputs of *ADV* and *ALG* are $T_{ADV} = (m+n)B$ and $E[T_{ALG}] = T_{ADV} - \sum_{i=1}^n l_i \leq (m+n)B - \sum_{i=1}^n \left(\frac{m-1}{m} \right)^i B$, giving the ratio

$$r = \frac{T_{ADV}}{E[T_{ALG}]} \geq \frac{(m+n)B}{(m+n)B - \sum_{i=1}^n \left(\frac{m-1}{m} \right)^i B} = \frac{m+n}{m+n - \sum_{i=1}^n \left(\frac{m-1}{m} \right)^i}.$$

The number of rounds n is to be chosen such that r is maximized, which is equivalent to the minimization of its reciprocal

$$\frac{m+n - \sum_{i=1}^n \left(\frac{m-1}{m} \right)^i}{m+n} = 1 - \frac{1}{m+n} \sum_{i=1}^n \left(\frac{m-1}{m} \right)^i.$$

Hence we want to maximize $\frac{1}{m+n} \sum_{i=1}^n \left(\frac{m-1}{m} \right)^i$. In order to analyze this ratio we consider a more general term $a_n = \frac{1}{m+n} \sum_{i=1}^n q^i$, where $0 < q < 1$. In the following two lemmas, we show that the sequence (a_n) first increases and then decreases from where we deduce a useful property for the sequence item where the maximum is achieved.

LEMMA 5. *If $a_{n+1} < a_n$, then $a_{n+2} < a_{n+1}$.*

Proof. Let $a_{n+1} < a_n$. Algebraic manipulations give $(m+n) \sum_{i=1}^{n+1} q^i < (m+n+1) \sum_{i=1}^n q^i$ and $(m+n)q^{n+1} < \sum_{i=1}^n q^i$. Using the latter inequality, we obtain

$$(m+n+1)q^{n+2} < (m+n)q^{n+1} + q^{n+2} < \sum_{i=1}^n q^i + q^{n+2} < \sum_{i=1}^{n+1} q^i.$$

This implies

$$(m+n+1) \sum_{i=1}^{n+1} q^i + (m+n+1)q^{n+2} < (m+n+1) \sum_{i=1}^{n+1} q^i + \sum_{i=1}^{n+1} q^i = (m+n+2) \sum_{i=1}^{n+1} q^i.$$

We conclude

$$(m + n + 1) \sum_{i=1}^{n+2} q^i < (m + n + 2) \sum_{i=1}^{n+1} q^i$$

and finally, as desired,

$$\frac{1}{m + n + 2} \sum_{i=1}^{n+2} q^i < \frac{1}{m + n + 1} \sum_{i=1}^{n+1} q^i. \quad \square$$

COROLLARY 1. *There exists a unique $n \in \mathbf{N}$ such that $a_{n-1} \leq a_n$ and $a_n > a_{n+1}$. For this n , there holds $a_n = \max_{r \in \mathbf{N}} a_r$.*

LEMMA 6. *There is a unique $n \in \mathbf{N}$ such that $q < \frac{a_n}{q^n} \leq 1$.*

Proof. By Corollary 1, there exists a unique $n \in \mathbf{N}$ such that $a_{n-1} \leq a_n$ and $a_n > a_{n+1}$. These two inequalities are equivalent to $\sum_{i=1}^{n-1} q^i \leq (m + n - 1)q^n$ and $\sum_{i=1}^n q^i > (m + n)q^{n+1}$. This is equivalent to $(m + n)q^{n+1} < \sum_{i=1}^n q^i \leq (m + n)q^n$. Dividing by q^n and $m + n$, we obtain the inequality to be proven. \square

COROLLARY 2. *$a_n = \max_{r \in \mathbf{N}} a_r$ if and only if $q < \frac{a_n}{q^n} \leq 1$.*

LEMMA 7. *If $q = (m - 1)/m$ and $a_{n(m)} = \max_{r \in \mathbf{N}} a_r$, then $\lim_{m \rightarrow \infty} n(m)/m = x$, where x is the unique positive solution of the equation $e^x = x + 2$.*

Proof. We fix m and look for n such that $a_n/q^n = 1$. Then, by Corollary 2, $n(m) = \lfloor n \rfloor$. Define \tilde{q} by $q\tilde{q} = 1$. We have $\tilde{q} - 1 = m/(m - 1) - 1 = 1/(m - 1)$ and $m = \tilde{q}/(\tilde{q} - 1)$. The equation $a_n/q^n = 1$ is equivalent to $\sum_{i=1}^n q^i = (m + n)q^n$, which in turn is equivalent to $\sum_{i=1}^n \tilde{q}^{-i} = (m + n)\tilde{q}^{-n}$. We obtain $\sum_{i=0}^{n-1} \tilde{q}^i = m + n$, and hence $\tilde{q}^n - 1 = (m + n)/(\tilde{q} - 1)$. Thus $\tilde{q}^n = (2m + n - 1)/(\tilde{q} - 1)$. Since $m = \tilde{q}/(\tilde{q} - 1)$, we have

$$\tilde{q}^n = \left(2 \frac{\tilde{q}}{\tilde{q} - 1} + n - 1\right) / \left(\frac{\tilde{q}}{\tilde{q} - 1} - 1\right) = 2\tilde{q} + (n - 1)(\tilde{q} - 1) = (\tilde{q} - 1)n + (\tilde{q} + 1).$$

Define α by $n = \alpha m$. We conclude that the inequality $a_n/q^n = 1$ is equivalent to $\tilde{q}^n = (\tilde{q} - 1)n + (\tilde{q} + 1)$, which in turn is equivalent to

$$\left(\left(\frac{m}{m - 1}\right)^m\right)^\alpha = \left(\frac{m}{m - 1} - 1\right) \alpha m + \left(\frac{m}{m - 1} + 1\right) = \frac{m}{m - 1} \alpha + \frac{2m - 1}{m - 1}.$$

For $m \rightarrow \infty$, this equation takes the form $e^\alpha = \alpha + 2$. \square

It remains to determine the competitive ratio r . We have

$$a_{n(m)} = \frac{\sum_{i=1}^{\lfloor \alpha_m m \rfloor} \left(\frac{m-1}{m}\right)^i}{m + \lfloor \alpha_m m \rfloor} \geq \frac{m - 1}{m} \frac{1 - \left(\frac{m-1}{m}\right)^{\alpha_m m - 1}}{1 - \frac{m-1}{m}} \frac{1}{m + \lfloor \alpha_m m \rfloor} \xrightarrow{m \rightarrow \infty} \frac{1 - e^{-\alpha}}{1 + \alpha} =: p.$$

For the ratio r , there holds

$$r \geq 1/(1 - p) = 1 / \left(1 - \frac{1 - e^{-\alpha}}{1 + \alpha}\right) = \frac{1 + \alpha}{1 + \alpha - 1 + e^{-\alpha}} = \frac{1 + \alpha}{1 + \alpha e^{-\alpha}} e^\alpha.$$

Since $e^\alpha = \alpha + 2$, we get

$$r = \frac{(1 + \alpha)(\alpha + 2)}{1 + \alpha(\alpha + 2)} = \frac{(1 + \alpha)(\alpha + 2)}{\alpha^2 + 2\alpha + 1} = \frac{(\alpha + 1)(\alpha + 2)}{(\alpha + 1)^2} = \frac{\alpha + 2}{\alpha + 1} = 1 + \frac{1}{\alpha + 1}.$$

Numerically solving equation $e^\alpha = \alpha + 2$ yields $\alpha \approx 1.1462$ and $r \approx 1.4659$. \square

3. Upper bounds. We first present our new algorithm *SGR* and then study the influence of resource augmentation.

3.1. A new semigreedy algorithm. We have seen in section 2.1 that any greedy algorithm is not better than 2-competitive. Nevertheless, it is the most reasonable decision to serve a queue that is (nearly) completely populated because, otherwise, the adversary sends packets to such a queue and the online algorithm loses almost all of them. In the request sequence from which we derive the lower bound of 2 for greedy algorithms, it is a crucial point that *GR* always serves the queue with the maximum load and ignores all other queues no matter how small the load difference is and causes the packet loss at the latter queues. Thus, the number of queues in which packets are lost increases. We shall now present a new algorithm *SGR* which embodies the greedy criterion but simultaneously tries to keep small the number of queues in which packet loss occurs. At queues that have already been fully populated, the greedy rule is relaxed and queues with less load are preferred. The risk of losing packets at the ignored queues that the adversary does not lose is low because the high load indicates that a lot of packets have already arrived there and, hence, the adversary presumably has a high load there as well. We now give a precise description of our algorithm.

ALGORITHM *SGR*. In each time step the algorithm executes the first rule that applies to the current buffer configuration.

1. If there is a queue buffering more than $\lfloor B/2 \rfloor$ packets, serve the queue currently having the maximum load.
2. If there is a queue, the hitherto maximum load of which is less than B , serve among these queues the one currently having the maximum load.
3. Serve the queue currently having the maximum load.

Ties are broken by choosing the queue with the smallest index. The hitherto maximum load is reset to 0 for all queues whenever all queues are unpopulated in *SGR*'s configuration.

In the remainder of this subsection we analyze *SGR*. Let σ be any packet arrival sequence. We divide σ into several phases, where phase P_1 starts at time $t = 0$. Phase P_i ends when *SGR* has completely cleared its buffers for the i th time, and phase P_{i+1} starts when the first packet arrives after the end of phase P_i . W.l.o.g. we can assume that at the beginning of each phase *ADV*'s buffers are empty as well because we can always postpone the start of the subsequent phase until this state is reached, thus only enlarging *ADV*'s throughput and hence the throughput ratio. So, the total throughput is given by the sum of the throughputs in the distinct phases if these were served separately. We derive that the ratio of the total throughputs cannot be worse than the throughput ratio in a single phase. Hence we can restrict ourselves to arrival sequences that terminate when *SGR* has completely cleared its buffers. Furthermore it suffices to consider sequences where, at any time step, if there is packet loss at queue i , either *SGR* or *ADV* loses packets. If both lose packets, then we can reduce the number of packets arriving at i by the minimum loss of any of the two algorithms.

For the analysis of *SGR* we introduce a new potential function. Let $l_{i,t,A}$ and $l_{i,t,S}$ be the load of queue i at time t in the configurations of *ADV* and *SGR*, respectively. We define the potential of the queue as $\Phi_{it} = (l_{i,t,A} - l_{i,t,S})_+$, where $x_+ = (x + |x|)/2$. The total potential at time t is given by $\Phi_t = \sum_{i=1}^m \Phi_{it}$. Let T be the time when *SGR* has completely emptied its buffers. Since both *ADV* and *SGR* transmit one packet in each time step until T , the potential Φ_T describes their throughput difference. By

$\Delta\Phi$ we denote the potential change $\Phi_t - \Phi_{t-1}$. We assume that during each time step, the arrival step precedes the transmission step.

LEMMA 8. *The potential Φ does not increase during the arrival step, but Φ decreases by the number of packets lost by ADV.*

Proof. Let p packets arrive at queue i at time step t . If neither *ADV* nor *SGR* loses any packet, there holds $l_{it,A} = l_{i,t-1,A} + p$, $l_{it,S} = l_{i,t-1,S} + p$, where we measure the load at $t-1$ after the transmission step. Hence $\Phi_{it} = (l_{it,A} - l_{it,S})_+ = ((l_{i,t-1,A} + p) - (l_{i,t-1,S} + p))_+ = (l_{i,t-1,A} - l_{i,t-1,S})_+ = \Phi_{i,t-1}$ and the potential does not change due to the packet arrival at queue i . As mentioned above we can restrict ourselves to arrival sequences, where either *ADV* or *SGR* loses packets whenever packet loss occurs. First we assume that *SGR* loses some packets. We derive $l_{i,t-1,A} < l_{i,t-1,S}$ and, hence, $\Phi_{i,t-1} = 0$. After the arrival of the p packets, there holds $l_{it,S} = B$ yielding $l_{it,A} \leq l_{it,S}$ and, hence, $\Phi_{it} = 0 = \Phi_{i,t-1}$. Since the single potentials do not change, neither does the total potential. Now we assume that *ADV* loses p_0 of the arriving p packets. Then $\Phi_{i,t-1} = (l_{i,t-1,A} - l_{i,t-1,S})_+ \geq p_0$. After the arrival of the p packets, there hold $l_{it,A} = B = l_{i,t-1,A} + (p - p_0)$ and $l_{it,S} = l_{i,t-1,S} + p$ yielding $(l_{it,A} - l_{it,S})_+ = (l_{i,t-1,A} + (p - p_0) - l_{i,t-1,S} - p)_+ = (l_{i,t-1,A} - l_{i,t-1,S} - p_0)_+ = (l_{i,t-1,A} - l_{i,t-1,S})_+ - p_0$ and, hence, $\Phi_{it} = \Phi_{i,t-1} - p_0$. \square

LEMMA 9. *During each transmission step the potential can change by at most 1.*

Proof. If *ADV* and *SGR* serve the same queue, the potential does not change at all. Let *ADV* serve queue i while *SGR* serves queue $j \neq i$. Let t^a and t be the moments after the arrival and the transmission steps, respectively. $\Phi_{it} = (l_{it,A} - l_{it,S})_+ = (l_{it^a,A} - l_{it^a,S})_+ = (l_{it^a,A} - l_{it^a,S} - 1)_+$ and $\Phi_{jt} = (l_{jt,A} - l_{jt,S})_+ = (l_{jt^a,A} - (l_{jt^a,S} - 1))_+ = (l_{jt^a,A} - l_{jt^a,S} + 1)_+$. Since $0 \leq (x+1)_+ - x_+ \leq 1$, we derive $-1 \leq \Phi_{it} - \Phi_{it^a} \leq 0$ and $0 \leq \Phi_{jt} - \Phi_{jt^a} \leq 1$. Hence there holds $-1 \leq \Phi_t - \Phi_{t^a} \leq 1$. \square

LEMMA 10. *During transmission step t the potential increases if and only if ADV serves a queue currently having potential 0 while SGR serves a queue having positive potential afterward.*

Proof. Let the potential increase by 1. Let *ADV* serve queue i while *SGR* serves queue $j \neq i$. As shown in Lemma 9, $-1 \leq \Delta\Phi_{it} \leq 0$ and $0 \leq \Delta\Phi_{jt} \leq 1$. Hence the potential increases by 1 if and only if $\Delta\Phi_{it} = 0$ and $\Delta\Phi_{jt} = 1$. On the one hand, if $\Phi_{i,t-1} > 0$, there holds $\Phi_{it} = \Phi_{i,t-1} - 1$ and, hence, $\Delta\Phi_{it} = -1$. On the other hand, if $\Phi_{jt} = 0$, there holds $\Phi_{j,t-1} = 0$ and, hence, $\Delta\Phi_{jt} = 0$. So, the potential increases if and only if $\Phi_{i,t-1} = 0$ and $\Phi_{jt} > 0$. \square

Before stating the next lemma, in the following three paragraphs, we define some new terms and introduce some further notation that will be used throughout the remainder of this section. If the potential increases, the queue served by *SGR* is called the *causal queue*, whereas the one served by *ADV* is termed *counter queue*. Whenever the potential increases, the situation of *ADV* worsens in such a way that *ADV* buffers more packets in queue j than *SGR* does. But, the same is true, interchanging their roles, for *SGR* and queue i . The difference in the number of packets held by *ADV* and *SGR* in queue i increases whenever Φ increases and *ADV* serves i . Hence, Φ measures the number of packets that *SGR* has already lost or could lose if *ADV* replenishes the corresponding queues. These replenishments are necessary to generate a larger throughput for *ADV*.

If t is a time step where the potential does not increase during packet transmission, we call t an *additional time step* and denote their number by T_0 . Hence, Φ increases $T - T_0$ times. Let T_{-1} be the number of time steps in which the potential decreases during packet transmission. Obviously, $T_{-1} \leq T_0$. Furthermore, we derive from

Lemma 8 that Φ decreases by L_0 during the arrival steps, where L_0 is the total number of packets lost by ADV . We have $\Phi_T = T - T_0 - T_{-1} - L_0$. Let T_A and T_S denote the throughputs of ADV and SGR , respectively. From $T_A = T_S + \Phi_T$ and $T_S = T$, we derive

$$(2) \quad \frac{T_A}{T_S} = \frac{T + \Phi_T}{T} \leq \frac{T + T - T_0 - T_{-1} - L_0}{T} = 2 - \frac{T_0 + T_{-1} + L_0}{T}.$$

In the following, we shall present buffer configurations, where the potential does not increase, and hence derive that $\frac{T_0 + T_{-1} + L_0}{T} \geq \frac{1}{9}$, establishing the $(2 - \frac{1}{9})$ -competitiveness of SGR .

The queues are divided into $B + 1$ sets Q^0, \dots, Q^B , where queue q is a member of Q^k if and only if k is the maximum load of q in SGR 's configuration while processing σ . In the following analysis we can ignore Q^0 , as no packets arrive there. Furthermore, depending on SGR 's configuration at q , we call q a Q_2 -queue if SGR currently buffers at least $\lfloor \frac{B}{2} \rfloor + 1$ packets in q . In case of SGR 's buffering at most $\lfloor \frac{B}{2} \rfloor$ packets in q , we call q a Q_{12} -queue if q has already had a total load of B packets, i.e., all buffers have been populated; otherwise we call it a Q_{11} -queue. By s_{11}, s_{12} , and s_2 we denote the number of time steps in which the potential increases while SGR serves a Q_{11} -, Q_{12} -, and Q_2 -queue, respectively. We partition s_{11} into s_{111} and s_{112} , where we assign an s_{11} time step to s_{111} if the counter queue is a Q_{11} -queue in SGR 's configuration, and to s_{112} , otherwise. Finally, let $b = \lfloor \frac{B}{2} \rfloor$, $L = \{1, \dots, b\}$, $R = \{b + 1, \dots, B\}$, and $R^* = R \setminus \{B\}$.

LEMMA 11. *The following inequality holds:*

$$L_0 + T_0 \geq \left(s_2 - (B - b) \sum_{k \in R} |Q^k| \right)_+.$$

Proof. Let $k \in R$ and $j \in Q^k$ be a queue that SGR 's serving of which has already increased s_2 at least $B - b$ times. We conclude that at least B packets have arrived at j because, at each of these time steps, j buffers more than b packets and one packet is transmitted at each such time step. For each further increase of s_2 , a buffer in queue j must have been repopulated due to a further packet arrival. If ADV can accept the arriving packet, ADV must have served j , too. In these time steps, ADV serves a queue having positive potential. Hence, they are additional. If ADV cannot accept the arriving packet, ADV has a further packet loss. Thus, s_2 can increase at most $B - b$ times at SGR 's serving j without additional time steps or packet loss for ADV , and each further such increase induces either an additional time step or another packet loss for ADV and, thus, increases by 1 either T_0 or L_0 . Since we have this result for all queues $j \in Q^k, k \in R$, there can be at most $(B - b) |\bigcup_{k \in R} Q^k| = (B - b) \sum_{k \in R} |Q^k|$ time steps at which s_2 increases without an increase of $L_0 + T_0$. We derive that if s_2 exceeds $(B - b) \sum_{k \in R} |Q^k|$, the amount of exceedence is a lower bound for $L_0 + T_0$. Since $L_0 + T_0$ is nonnegative, we can strengthen our result by adding the $(\cdot)_+$ operator. \square

LEMMA 12. *The following inequality holds:*

$$T_0 \geq (s_{111} - b|Q^B|)_+ + s_{112} + s_{12} + (B - 2b)|Q^B| - T_{-1}.$$

Proof. We investigate s_{11} and s_{12} increases and focus on steps in which queue i is counter queue. We first study the case when i is a Q_{11} -queue and then consider the setting when i is a Q_{12} -queue.

We first observe that if i is a Q_{11} -queue, then by the definition of SGR the potential increase cannot be one assigned to s_{12} . Suppose that i is used n_i times as a counter queue for an s_{11} increase while i is a Q_{11} queue itself. We denote these time steps by $t_1 < \dots < t_{n_i}$. First, we assume that $i \in Q^k, k < B$. In this case, SGR has no packet loss in i . Due to Lemma 10, ADV buffers at most as many packets in i as SGR before the transmission steps at t_1, \dots, t_{n_i} . Since each packet is transmitted only once, there exist time steps $t'_1 < \dots < t'_{n_i}$, with $t'_j > t_j, 1 \leq j \leq n_i$, where SGR serves i while ADV does not and SGR buffers more packets than ADV in i . These time steps are additional. Now, assume that $i \in Q^B$ and that SGR buffers B packets in i for the first time at time t . Since i cannot be a Q_{11} -queue after t , we derive that $t_{n_i} < t$. At each $t_j, 1 \leq j \leq n_i$, SGR buffers at most b packets in i . We claim that there exist time steps $t'_1 < \dots < t'_{n_i-b} < t$, with $t'_j > t_j, 1 \leq j \leq n_i - b$, where SGR serves queue i while ADV does not and SGR buffers more packets than ADV in queue i . These time steps are additional. To see the claim, take the time steps t_j in increasing order and match each one with the next unmatched time step $t'_j > t_j$ at which SGR serves i . Clearly, the unmatched t_j form a consecutive subsequence of t_1, \dots, t_{n_i} that includes t_{n_i} . Each unmatched time step corresponds to an increase of 1 in the buffer population difference between SGR and ADV . If more than b time steps were unmatched, then SGR would buffer more than b packets in i at time t_{n_i} , contradicting the fact that i is a Q_{12} -queue at that time. Hence, at most the b time steps $t_j, n_i - b < j \leq n_i$, are not matched. Combining the arguments of this paragraph, we derive that $T_0 \geq (s_{111} - b|Q^B|)_+$.

Now, we assume that at time t' , queue i is a Q_{12} -queue and used as a counter queue for an s_{11} or s_{12} increase. Then, there holds $i \in Q^B$ and $t' \geq t$. By Lemma 10, in both ADV 's and SGR 's configurations, only the L -part is populated in queue i . If i is used as a counter queue during $n'_i > b$ increases of s_{11} or s_{12} and it is a Q_{12} -queue at all the respective time steps, then, as above, we can prove that there are at least $n'_i - b$ time steps in which SGR serves queue i while ADV does not and SGR buffers more packets than ADV in queue i . These time steps are additional. Moreover, these time steps are different from those identified in the last paragraph because they occur after t .

Next, we consider the first use of i as a counter queue for an s_{11} or s_{12} increase when i is a Q_{12} -queue. Consider the first $B - b$ time steps after t satisfying one of the following properties: (a) both SGR and ADV serve i ; (b) only the algorithm currently having the larger buffer population in i serves i . These time steps exist and all occur before the first use of i being a Q_{12} -queue because SGR serves i at least $B - b$ times during that time window. The $B - b$ time steps are additional. However, they need not be distinct for the various queues i and i' and need not be distinct from the additional time steps identified so far. If a time step is counted twice, ADV serves i having a larger population than SGR in that queue, while SGR serves i' having a larger population in there. Note that the potential drops during packet transmission in this case. In order not to count one time step twice when summing up over all queues, we diminish this amount of time steps by T_{-1} . Since $n'_i - b + B - b = n'_i + (B - 2b)$, combining the arguments of the three paragraphs, we obtain the lemma. \square

While the two lemmas above cover the case that there are many time steps in which the potential increases, the following one considers the opposite situation.

LEMMA 13. *The following inequality holds:*

$$T_0 \geq T_{0,11} + T_{0,12} + T_{0,2},$$

where

$$\begin{aligned} T_{0,11} &= \left(\sum_{k \in L} k|Q^k| + \sum_{k \in R^*} b|Q^k| - s_{11} \right)_+, \\ T_{0,12} &= (b|Q^B| - s_{12})_+, \\ T_{0,2} &= \left(\sum_{k \in R} (k - b)|Q^k| - s_2 \right)_+. \end{aligned}$$

Proof. *SGR* must serve each buffer cell that is populated at least once. If $k \in L$ and $i \in Q^k$, in *SGR*'s configuration, the populated part consists only of the k first cells and i is always a Q_{11} -queue and, hence, can only cause s_{11} increases. If $k \in R^*$ and $i \in Q^k$, in *SGR*'s configuration, i is always a Q_{11} -queue when the j th cell, $1 \leq j \leq b$, becomes unpopulated and, hence, can only cause s_{11} increases at these time steps. If $i \in Q^B$, in *SGR*'s configuration, i is always a Q_{12} -queue when the j th cell, $1 \leq j \leq b$, becomes unpopulated after the buffer has once been populated by B packets and can only cause s_{12} increases afterward. If $k \in R$ and $i \in Q^k$, in *SGR*'s configuration, the populated R -part consists of $k - b$ cells. There must be additional time steps if the potential increases less often than the number of populated buffer cells. \square

LEMMA 14. *If $B \geq 2$, the following inequalities hold:*

$$\begin{aligned} T_0 + T_{-1} &\geq s_{12}, \\ 2T_0 + T_{-1} &\geq s_{11} + (B - 2b)|Q^B|, \\ L_0 + 5T_0 + 2T_{-1} &\geq s_2 - (B - 2b) \sum_{k \in R^*} |Q^k|. \end{aligned}$$

Proof. The first inequality is an immediate result of Lemma 12. Summing up the inequalities of Lemmas 12 and 13 yields

$$\begin{aligned} 2T_0 + T_{-1} &\geq (s_{111} - b|Q^B|)_+ + s_{112} + s_{12} + (B - 2b)|Q^B| + (b|Q^B| - s_{12})_+ \\ &\geq s_{111} + s_{112} + (-b + (B - 2b) + b)|Q^B| \\ &= s_{11} + (B - 2b)|Q^B|, \end{aligned}$$

establishing the second inequality. By the summation of the inequalities of Lemmas 11, 12, and 13, we get

$$\begin{aligned} L_0 + 3T_0 + T_{-1} &\geq \left(s_2 - (B - b) \sum_{k \in R} |Q^k| \right)_+ + s_{112} + s_{12} \\ &\quad + (B - 2b)|Q^B| + (b|Q^B| - s_{12})_+ \\ &\quad + \left(\sum_{k \in L} k|Q^k| + \sum_{k \in R^*} b|Q^k| - s_{11} \right)_+ \\ &\geq s_2 - s_{111} - (B - 2b) \sum_{k \in R^*} |Q^k|. \end{aligned}$$

The addition of the second inequality of the lemma yields $L_0 + 5T_0 + 2T_{-1} \geq s_2 - (B - 2b) \sum_{k \in R^*} |Q^k|$. \square

THEOREM 5. *SGR achieves a competitive ratio of 17/9.*

Proof. Let β denote $(B - 2b) \sum_{k \in R^*} |Q^k|$. If $i \in Q^k$ and $k \in R^*$, at least $b + 1$ packets arrive at i . Hence $T \geq (b + 1)\beta$, and, thus, $\frac{\beta}{T} \leq \frac{1}{b+1}$. The summation of the three inequalities in Lemma 14 yields $L_0 + 8T_0 + 4T_{-1} \geq s_{11} + s_{12} + s_2 - \beta$. We derive

$$L_0 + 9T_0 + 4T_{-1} \geq s_{11} + s_{12} + s_2 + T_0 - \beta = T - \beta$$

and, hence, $L_0 + T_0 + T_{-1} \geq \frac{1}{9}L_0 + T_0 + \frac{4}{9}T_{-1} \geq \frac{T-\beta}{9}$, yielding

$$(L_0 + T_0 + T_{-1})/(T - \beta) \geq 1/9.$$

If B is even, then $\beta = 0$ and $(L_0 + T_0 + T_{-1})/T \geq 1/9$, and by (2) the theorem follows. If B is odd, $\frac{L_0 + T_0 + T_{-1}}{T} = \frac{L_0 + T_0 + T_{-1}}{T - \beta} (1 - \frac{\beta}{T}) \geq \frac{1}{9} (1 - \frac{1}{b+1}) = \frac{1}{9} (1 - \frac{2}{B+1})$. Let $\delta_B = \frac{2}{B+1}$. Then $\lim_{B \rightarrow \infty} \delta_B = 0$ and we derive by (2) that

$$T_A/T_S \leq 2 - (1 - \delta_B)/9 = 17/9 + \delta_B/9 \rightarrow 17/9 \text{ as } B \rightarrow \infty. \quad \square$$

We can strengthen our analysis and show that, for $B = 2$, *SGR* is optimal.

THEOREM 6. *For $B = 2$, *SGR* achieves a competitive ratio of $13/7$.*

Proof. If we put $B = 2$ into the inequalities of Lemmas 11, 12, and 13, they take the following forms: $L_0 + T_0 \geq (s_2 - |Q^2|)_+$, $T_0 \geq (s_{111} - |Q^2|)_+ + s_{112} + s_{12} - T_{-1}$, and $T_0 \geq (|Q^1| - s_{11})_+ + (|Q^2| - s_{12})_+ + (|Q^2| - s_2)_+$, from which we deduce the corresponding inequalities of Lemma 14 for $B = 2$: $T_0 + T_{-1} \geq s_{12}$ and $2T_0 + T_{-1} \geq s_{11}$. Summing up the three inequalities in Lemmas 11, 12, and 13, we can strengthen the third inequality of Lemma 14: $L_0 + 3T_0 + T_{-1} \geq (s_2 - |Q^2|)_+ + s_{12} + (|Q^2| - s_{12})_+ \geq s_2 - |Q^2| + s_{12} + |Q^2| - s_{12} = s_2$. Since $5T_0$ is replaced by $3T_0$ in the left-hand side, we get a competitive factor of $\frac{13}{7}$ instead of $\frac{17}{9}$ in Theorem 5. \square

3.2. Resource augmentation. We first study the case that an online algorithm is granted additional buffer space. Then we consider different transmission rates.

3.2.1. Additional buffer space. Let *ADV* and *ALG* have buffers of size B and $B + A$ for each queue, respectively. We denote the ratio of the additional buffer space A and the standard buffer space B by c . Recall that an algorithm is called work-conserving if in any time step it serves some nonempty queue, provided such nonempty queues exist.

THEOREM 7. *Every work-conserving online algorithm *ALG* having additional buffer space $A = cB$ per queue is $(\frac{c+2}{c+1})$ -competitive.*

Proof. Let σ be any arrival sequence. As in the analysis of *SGR*, we can restrict ourselves to arrival sequences that terminate when *ALG* has completely cleared its buffer. Furthermore, we assume that in the case of packet loss at a queue either *ADV* or *ALG* loses packets. If there is no packet loss at all or if only *ADV* loses packets, then the throughput of *ALG* is at least as large as that of *ADV* and our claim holds. Hence we may assume that *ALG* loses packets. We partition the queues into two disjoint sets Q^ℓ and Q^o , where Q^ℓ comprises exactly those in which *ALG* loses packets when serving σ . The difference in the total packet losses corresponds to *ADV*'s buffer population at the time when *ALG*'s buffers become empty. Let T_{ADV} and T_{ALG} denote the throughputs of *ADV* and *ALG*, respectively. By L_{ADV} and L_{ALG} we denote the losses of *ADV* and *ALG*, respectively. We derive

$$T_{ADV} - T_{ALG} = L_{ALG} - L_{ADV} \leq |Q^\ell|B.$$

Furthermore, the total throughput is given by the throughputs in Q^o and Q^ℓ , denoted by $T_{ADV}^o, T_{ADV}^\ell, T_{ALG}^o$, and T_{ALG}^ℓ , respectively. So $T_{ADV} = T_{ADV}^o + T_{ADV}^\ell$ and

$T_{ALG} = T_{ALG}^o + T_{ALG}^\ell$. Since there occurred packet loss in the queues in Q^ℓ , at least $A + B = (1 + c)B$ packets must have arrived there, all of which could be accepted by ALG . Hence there holds

$$T_{ALG}^\ell \geq (1 + c)|Q^\ell|B$$

yielding

$$\begin{aligned} \frac{T_{ADV}}{T_{ALG}} &= \frac{T_{ALG} + L_{ALG} - L_{ADV}}{T_{ALG}} = 1 + \frac{L_{ALG} - L_{ADV}}{T_{ALG}} = 1 + \frac{L_{ALG} - L_{ADV}}{T_{ALG}^o + T_{ALG}^\ell} \\ &\leq 1 + \frac{L_{ALG} - L_{ADV}}{T_{ALG}^\ell} \leq 1 + \frac{|Q^\ell|B}{(1 + c)|Q^\ell|B} = 1 + \frac{1}{1 + c} = \frac{c + 2}{c + 1}. \quad \square \end{aligned}$$

THEOREM 8. *For any greedy algorithm GR , the upper bound of $\frac{c+2}{c+1}$ is tight for all B and all $A = cB$.*

Proof. For all values of A and B we construct an instance with a throughput ratio arbitrarily close to $\frac{c+2}{c+1}$. First, we assume that both ADV and GR have buffers of size $B + A$. From the previous section we know that we can construct a staircase sequence such that ADV has a throughput of about $2m(B + A)$, whereas GR has a throughput of only $m(B + A)$ packets. While the staircase is built up in GR , ADV always serves the corresponding queues such that they are empty in ADV 's configuration. This results in the fact that ADV can accept additional $B + A$ packets in these queues. Although—now—the buffer of ADV is smaller than the one of GR , ADV can build up a staircase of level $B + A$ in GR 's configuration. The difference consists only of ADV 's not being able to accept $B + A$, but only B additional packets per queue, while GR cannot accept any of them. Hence, GR accepts $B + A$ packets per queue, whereas ADV accepts $B + A + B = 2B + A$ packets per queue. This results in the following throughput ratio:

$$\frac{T_{ADV}}{T_{GR}} = \frac{2B + A}{B + A} = \frac{2B + cB}{B + cB} = \frac{c + 2}{c + 1}. \quad \square$$

3.2.2. Increased transmission rate. Now we assume that ADV and ALG have the same buffer sizes, but ALG can transmit k packets per time step while ADV is still able to transmit only one packet per time step.

THEOREM 9. *Every work-conserving online algorithm ALG is $(1 + \frac{1}{k})$ -competitive if its transmission rate is the k -fold of the adversary's one.*

Proof. As usual, we restrict ourselves to sequences which terminate when ALG has completely emptied its buffers and assume that in the case of packet loss at a queue either ADV or ALG loses packets. Let $l_{it,ADV}$ and $l_{it,ALG}$ be the loads (i.e., the numbers of packets) in queue i at time t in the configurations of ADV and ALG , respectively. By $L_{it,ADV}$ and $L_{it,ALG}$ we denote the numbers of packets ADV and ALG lose at queue i at time t , respectively. Let T_{ADV} , L_{ADV} , T_{ALG} , and L_{ALG} denote the throughputs and losses of ADV and ALG , respectively. For each queue i we define a potential function $\Phi_{it} = (l_{it,ALG} - l_{it,ADV})_+$ which sums up to $\Phi_t = \sum_{i=1}^m \Phi_{it}$. At time t , ALG can lose packets in queue i only if $\Phi_{it} > 0$. Then $\Delta\Phi_{it} = -L_{it,ALG}$. During each transmission step, Φ_{it} can increase only if ADV serves a queue not served by ALG . But if so, ALG transmits k packets from other queues, hence $\frac{1}{k}\Delta T_{t,ALG} \geq \Delta\Phi_{it}$. So we derive

$$\Delta\Phi_t \leq \frac{1}{k}\Delta T_{t,ALG} - L_{t,ALG}.$$

Let τ denote the time step when ALG 's buffers become empty. Since $\Phi_0 = 0 = \Phi_\tau$, it is the case that $0 \leq \frac{T_{ALG}}{k} - L_{ALG}$, hence $L_{ALG} \leq \frac{T_{ALG}}{k}$, giving us the following throughput ratio:

$$\frac{T_{ADV}}{T_{GR}} = \frac{T_{ALG} + (L_{ALG} - L_{ADV})}{T_{ALG}} \leq \frac{T_{ALG} + L_{ALG}}{T_{ALG}} \leq 1 + \frac{1}{k}. \quad \square$$

THEOREM 10. *If $B = 1$, every work-conserving online algorithm ALG with transmission rate greater than that of the adversary by a factor of k has a competitive ratio of at least $1 + 1/k$.*

Proof. Let there be m queues, each of them populated at the beginning. While ALG serves $k \lfloor \frac{m}{k+1} \rfloor$ queues, ADV serves other $\lfloor \frac{m}{k+1} \rfloor$ ones. At time $\lfloor \frac{m}{k+1} \rfloor$, one packet arrives at each of the latter queues. ADV accepts all new packets, whereas ALG must reject of all of them. This pattern is repeated on the queues still populated by ALG , resulting in $\lfloor (\frac{1}{k+1})^2 m \rfloor$ new packets. We iterate this procedure on $\lfloor (\frac{1}{k+1})^3 m \rfloor, \lfloor (\frac{1}{k+1})^4 m \rfloor, \dots$ queues not served yet by ALG until the block size reaches 1. Since ALG accepts only the m initial packets and ADV can accept all the other packets as well, we get

$$\frac{T_{ADV}}{T_{ALG}} = \frac{\sum_{j=0}^{\infty} \lfloor (\frac{1}{k+1})^j m \rfloor}{m}$$

and this expression tends to

$$\frac{\sum_{j=0}^{\infty} (\frac{1}{k+1})^j m}{m} = 1 / \left(1 - \frac{1}{k+1} \right) = 1 + \frac{1}{k}$$

as m goes to infinity. \square

4. An optimal offline algorithm. We present an optimal offline algorithm for our problem.

ALGORITHM SFOD (SHORTEST FORWARD OVERFLOW DISTANCE). If there can no longer be any buffer overflow, serve the queues in an arbitrary order. Otherwise select in each transmission step t a queue in which the next overflow would occur if none of the queues were served. More precisely, let $l_{i,t}$ and $\sigma_{i,t}$ denote the length of queue i at time t and the number of packets arriving at queue i at time t , respectively. At time t , determine the earliest time step $t_0 > t$ such that $l_{i,t} + \sum_{\tau=t+1}^{t_0-1} \sigma_{i,\tau} \leq B$ for all i and $l_{i,t} + \sum_{\tau=t+1}^{t_0} \sigma_{i,\tau} > B$ for some queue i . If t_0 exists, select queue i ; otherwise serve any queue.

The algorithm can be implemented so that it runs in linear time.

THEOREM 11. *SFOD is an optimal offline algorithm.*

Proof. We prove the following two statements, which imply the theorem.

1. For each arrival sequence σ there exists an optimal schedule satisfying the *SFOD* rule.
2. Each schedule satisfying the *SFOD* rule is optimal.

We first prove statement 1. Let σ be any arrival sequence and let S be an optimal schedule for σ . We show that we can convert S into an *SFOD* schedule S' without decreasing the throughput. If S satisfies the *SFOD* property, then there is nothing to show. Now assume that there exists a time step t in which S does not satisfy *SFOD*. Let S serve queue i at time t and let queue j be a queue in which the next buffer overflow would occur after t . S' is a copy of S , but at time t , S' serves queue j

instead of i . Let t_0 denote the time step in which the next overflow in queue j would occur if it were no longer served. Due to the *SFOD* rule, the next overflow in queue j would occur before the next overflow in queue i . If S serves queue j at time t' with $t < t' < t_0$, S' serves queue i at time t' . Then at time t' both S and S' are again in the same configuration and their throughputs are the same. If S does not serve queue j until t_0 , the buffer will overflow, and S will lose one packet more than S' will. If there is an overflow at queue i later on, S' will lose one packet more than S will. Then, the configurations of S and S' are the same, again, and yield the same throughputs. Hence, in either case, we get $T_{S'} \geq T_S$. We can repeatedly apply this procedure until we obtain a schedule S' satisfying the *SFOD* rule. The procedure terminates because σ is finite.

We next prove statement 2. We derive from statement 1 that there is an optimal schedule \hat{S} satisfying the *SFOD* rule. Let S be any *SFOD* schedule. If there is a packet loss in S at time t , \hat{S} must lose the same number of packets because in both schedules queues have been served, where the next overflow was nearest in the future. Hence there can be a difference between S and \hat{S} only if there are several queues whose next overflows occur at the same time. If packet loss is inevitable at t , S and \hat{S} lose the same number of packets at t because they served the queues concerned as often as possible. \square

5. Conclusions and open problems. In this paper we have studied the problem of maximizing the throughput of unit-value packets at a switch with m input buffers. We have developed improved upper and lower bounds on the competitive performance of online algorithms. In particular, we have devised a strategy, called *semigreedy*, that achieves a competitiveness of $\frac{17}{9}$ and is the first deterministic algorithm that beats the trivial upper bound of 2. We remark here that after the publication of our preliminary work, improved deterministic and randomized algorithms were developed in [3, 16].

An important problem is to determine tight upper and lower bounds on the performance of deterministic and randomized algorithms. Furthermore, it is interesting to study scenarios in which data packets have values and we wish to maximize the total value of transmitted packets. As mentioned in the introduction, Azar and Richter [4] gave a general technique that transforms any c -competitive algorithm for a single queue into a $2c$ -competitive algorithm for multiqueue systems. As a result, they derived competitive algorithms for various settings such as preemptive and nonpreemptive models with both 2 values and an arbitrary number of values. It is conceivable that improved solutions are possible by investigating the respective settings directly.

Acknowledgments. We thank two anonymous referees for their helpful comments which improved the presentation of the paper.

REFERENCES

- [1] W. AIELLO, Y. MANSOUR, S. RAJAGOPOLAN, AND A. ROSÉN, *Competitive queue policies for differentiated services*, J. Algorithms, 55 (2005), pp. 113–141.
- [2] N. ANDELMAN, Y. MANSOUR, AND A. ZHU, *Competitive queueing policies for QoS switches*, in Proceedings of the 14th Annual ACM-SIAM Symposium on Discrete Algorithms, SIAM, Philadelphia, ACM, New York, 2003, pp. 761–770.
- [3] Y. AZAR AND A. LITICHESKEY, *Maximizing throughput in multi-queue switches*, in Proceedings of the 12th Annual European Symposium on Algorithms, Lecture Notes in Comput. Sci. 3221, Springer, Berlin, New York, 2004, pp. 53–64.

- [4] Y. AZAR AND Y. RICHTER, *Management of multi-queue switches in QoS networks*, in Proceedings of the 35th Annual ACM Symposium on Theory of Computing, ACM, New York, 2003, pp. 82–89.
- [5] Y. AZAR AND Y. RICHTER, *The zero-one principle for switching networks*, in Proceedings of the 36th ACM Symposium on Theory of Computing, ACM, New York, 2004, pp. 64–71.
- [6] Y. AZAR AND Y. RICHTER, *An improved algorithm for CIOQ switches*, in Proceedings of the 12th Annual European Symposium on Algorithms, Lecture Notes in Comput. Sci. 3221, Springer, Berlin, New York, 2004, pp. 65–76.
- [7] A. AZIZ, A. PRAKASH, AND V. RAMACHANDRAN, *A new optimal scheduler for switch-memory-switch routers*, in Proceedings of the 15th Annual ACM Symposium on Parallelism in Algorithms and Architectures, ACM, New York, 2003, pp. 343–352.
- [8] N. BANSAL, L. K. FLEISCHER, T. KIMBREL, M. MAHDIAN, B. SCHIEBER, AND M. SVIRIDENKO, *Further improvements in competitive guarantees for QoS buffering*, in Proceedings of the 31st International Colloquium on Automata, Languages, and Programming, Lecture Notes in Comput. Sci. 3142, Springer, Berlin, New York, 2004, pp. 196–207.
- [9] A. BAR-NOY, A. FREUND, S. LANDA, AND J. NAOR, *Competitive on-line switching policies*, *Algorithmica*, 36 (2003), pp. 225–247.
- [10] R. FLEISCHER AND H. KOGA, *Balanced scheduling toward loss-free packet queuing and delay fairness*, *Algorithmica*, 38 (2004), pp. 363–376.
- [11] E. L. HAHNE, A. KESSELMAN, AND Y. MANSOUR, *Competitive buffer management for shared-memory switches*, in Proceedings of the 13th Annual ACM Symposium on Parallel Algorithms and Architectures, ACM, New York, 2001, pp. 53–58.
- [12] A. KESSELMAN, Z. LOTKER, Y. MANSOUR, B. PATT-SHAMIR, B. SCHIEBER, AND M. SVIRIDENKO, *Buffer overflow management in QoS switches*, *SIAM J. Comput.*, 33 (2004), pp. 563–583.
- [13] A. KESSELMAN AND Y. MANSOUR, *Loss-bounded analysis for differentiated services*, *J. Algorithms*, 46 (2003), pp. 79–95.
- [14] A. KESSELMAN, Y. MANSOUR, AND R. VAN STEE, *Improved competitive guarantees for QoS buffering*, in Proceedings of the 11th Annual European Symposium on Algorithms, Lecture Notes in Comput. Sci. 2832, Springer, Berlin, 2003, pp. 361–372.
- [15] A. KESSELMAN AND A. ROSÉN, *Scheduling policies for CIOQ switches*, in Proceedings of the 15th Annual ACM Symposium on Parallelism in Algorithms and Architectures, ACM, New York, 2003, pp. 353–361.
- [16] M. SCHMIDT, *Packet buffering: Randomization beats deterministic algorithms*, in Proceedings of the 22nd Annual Symposium on Theoretical Aspects of Computer Science, Lecture Notes in Comput. Sci. 3404, Springer, Berlin, 2005, pp. 293–304.
- [17] D. D. SLEATOR AND R. E. TARJAN, *Amortized efficiency of list update and paging rules*, *Comm. ACM*, 28 (1985), pp. 202–208.

ANALYSIS OF LINK REVERSAL ROUTING ALGORITHMS*

COSTAS BUSCH[†] AND SRIKANTA TIRTHAPURA[‡]

Abstract. Link reversal algorithms provide a simple mechanism for routing in communication networks whose topology is frequently changing, such as in mobile ad hoc networks. A link reversal algorithm routes by imposing a direction on each network link such that the resulting graph is a destination oriented DAG. Whenever a node loses routes to the destination, it reacts by reversing some (or all) of its incident links. Link reversal algorithms have been studied experimentally and have been used in practical routing algorithms, including TORA [V. D. Park and M. S. Corson, A highly adaptive distributed routing algorithm for mobile wireless networks, in *Proc. INFOCOM*, IEEE, Los Alamitos, CA, 1997, pp. 1405–1413].

This paper presents the first formal performance analysis of link reversal algorithms. We study these algorithms in terms of *work* (number of node reversals) and the *time* needed until the network stabilizes to a state in which all the routes are reestablished. We focus on the *full reversal* algorithm and the *partial reversal* algorithm, both due to Gafni and Bertsekas [*IEEE Trans. Comm.*, 29 (1981), pp. 11–18]; the first algorithm is simpler, while the latter has been found to be more efficient for typical cases. Our results are as follows:

- The full reversal algorithm requires $O(n^2)$ work and time, where n is the number of nodes that have lost routes to the destination. This bound is tight in the worst case.
- The partial reversal algorithm requires $O(n \cdot a^* + n^2)$ work and time, where a^* is a nonnegative integral function of the initial state of the network. Further, for every nonnegative integer α , there exists a network and an initial state with $a^* = \alpha$, and with n nodes that have lost their paths to the destination, such that the partial reversal algorithm requires $\Omega(n \cdot a^* + n^2)$ work and time.
- There is an inherent lower bound on the worst-case performance of link reversal algorithms. There exist networks such that for *every* deterministic link reversal algorithm, there are initial states that require $\Omega(n^2)$ work and time to stabilize. Therefore, surprisingly, the full reversal algorithm is asymptotically optimal in the worst case, while the partial reversal algorithm is not, since a^* can be arbitrarily larger than n .

Key words. link reversal routing, wireless networks, ad hoc networks, fault tolerance, self stabilization

AMS subject classifications. 68W15, 68W40, 68Q17, 68Q25, 68Q85

DOI. 10.1137/S0097539704443598

1. Introduction. A mobile ad hoc network is a temporary interconnection network of mobile wireless nodes without a fixed infrastructure. The attractive feature of such a network is the ease with which one can construct it: there is no physical setup needed at all. If mobile nodes come within the wireless range of each other, then they will be able to communicate. More significant, even if two mobile nodes aren't within the wireless range of each other, they might still be able to communicate through a multihop path. However, the lack of a fixed infrastructure makes routing between nodes a hard problem. Since nodes are moving, the underlying communication graph is changing, and the nodes have to adapt quickly to such changes and reestablish their routes.

*Received by the editors May 8, 2004; accepted for publication (in revised form) May 12, 2005; published electronically October 7, 2005. A preliminary version of this paper has appeared in [2].

<http://www.siam.org/journals/sicomp/35-2/44359.html>

[†]Department of Computer Science, Rensselaer Polytechnic Institute, 110 8th Street, Troy, NY 12180 (buschc@cs.rpi.edu).

[‡]Department of Electrical and Computer Engineering, Iowa State University, 2215 Coover Hall, Ames, IA 50011 (snt@iastate.edu).

1.1. Link reversal. *Link reversal* routing algorithms [12, Chapter 8] are adaptive, self-stabilizing, distributed algorithms used for routing in mobile ad hoc networks. The first link reversal algorithms are due to Gafni and Bertsekas [7]. Link reversal is the basis of the temporally ordered routing algorithm TORA [10], and has also been used in the design of leader election algorithms for mobile ad hoc networks [9]. Link reversal routing is best suited for networks in which the rate of topological changes is high enough to rule out algorithms based on shortest paths, but not so high as to make flooding the only alternative.

Consider the graph representing the network, where the vertices are the wireless nodes, and each node has a link with each other node within its transmission radius. We assume that this underlying graph is undirected; i.e., the communication links are all bidirectional. Link reversal algorithms route on this graph by assigning directions to different links, hence converting it to a directed graph. A directed graph G is said to be connected if the underlying undirected graph of G (formed upon erasing the directions on the edges of G) is connected.

DEFINITION 1.1. *A connected directed acyclic graph with a single destination node is said to be destination oriented iff every directed path in the graph leads to the destination.*

For a given destination node, the link reversal algorithms assign directions to the links of this graph such that the resulting directed graph is a destination oriented directed acyclic graph (see Figure 1). Routing on a destination oriented network is easy: when a node receives a packet, it forwards the packet on *any* outgoing link, and the packet will eventually reach the destination.¹

The task of the link reversal algorithm is to create and maintain the routes to the destination. When two nodes move out of each other's range, the link between them is destroyed, and some nodes might lose their routes to the destination. The routing algorithm reacts by performing *link reversals* (i.e., reorienting some of the edges) so that the resulting directed graph is again destination oriented. In particular, when a node finds that it has become a *sink* (has lost all of its outgoing links), then the node reacts by reversing the directions of some or all of its incoming links. The link reversals due to one node may cause adjacent nodes to perform reversals, and in this way, the reversals propagate in the network until the routes to the destination are reestablished.

Gafni and Bertsekas [7] describe a general family of link reversal algorithms and present two particular algorithms: the *full reversal* algorithm and the *partial reversal* algorithm (referred to as the GB algorithms in the rest of this paper). In the full reversal algorithm, when a node becomes a sink, it reverses the directions of all of its incident links. In the partial reversal algorithm, the sink reverses the directions of only those incident links that have not been recently reversed by adjacent nodes (a detailed description appears in the following section). The full reversal algorithm is simpler to implement, but the partial reversal algorithm may need fewer link reversals in some cases. Gafni and Bertsekas show that when link failures occur, these algorithms *eventually* converge to a destination oriented graph. However, it was not known how many reversals the nodes performed, or how much time it would take till convergence.

1.2. Performance of link reversal. We present the first formal performance analysis of link reversal routing algorithms. We give tight upper and lower bounds

¹If there are multiple destinations in the network, then there is a separate directed graph for each destination; here, we will assume for simplicity that there is only one destination.

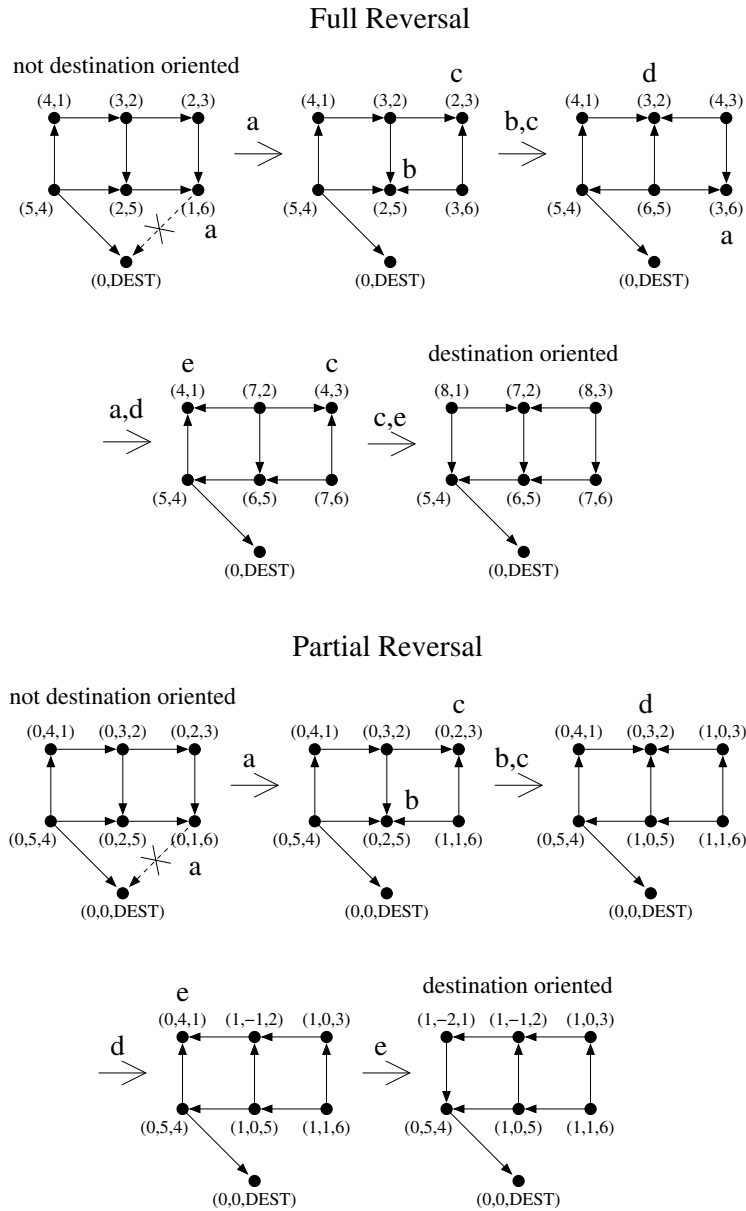


FIG. 1. Sample executions of the GB full and partial reversal algorithms. Each transition is labeled with the nodes that reverse.

on the performance of the full and partial reversal algorithms. We also show a lower bound on the performance of *any* deterministic link reversal algorithm. Surprisingly, from the perspective of worst-case performance, the full reversal algorithm is asymptotically optimal while the partial reversal algorithm is not.

Our setting for analysis is as follows. Suppose topological changes occur in the network, driving the system to a state in which some nodes have lost their paths to the destination. This is called the *initial state* of the network. If there are no further

topological changes, the network is said to have *stabilized* when it again becomes destination oriented. We analyze two metrics:

Work: The number of *node reversals* until stabilization; a node reversal is the action of a sink reversing some or all of its adjacent links. This is a measure of the power and computational resources consumed by the algorithm in reacting to topological changes.

Time: The number of parallel steps until stabilization, which is a measure of the speed in reacting to topological changes. We model reversals so that each reversal requires one time step, and reversals may occur simultaneously whenever possible.

Reversals are implemented using *heights*. A reversal algorithm assigns a height to every node in the network. The link between adjacent nodes is directed from the node of greater height to the node of lesser height. A sink performs a reversal by increasing its height by a suitable amount. This will reverse the direction of some or all of its incident links. We consider *deterministic* link reversal algorithms, in which a sink increases its height according to some deterministic function of its own height and the heights of the adjacent nodes. The GB link reversal algorithms are deterministic.

In the analysis, we separate the nodes into bad and good. A node is *bad* if there is no route from the node to the destination. Any other node, including the destination, is *good*. Note that a bad node is not necessarily a sink. We present results for the following algorithms.

Full reversal algorithm. For the full reversal algorithm, we show that when started from an initial state with n bad nodes, the work and time needed to stabilize is $O(n^2)$. This bound is tight. We show that there are networks with initial states which require $\Omega(n^2)$ time for stabilization.

Our result for full reversal is actually stronger. For any network, we present a decomposition of the bad nodes in the initial state into *layers*, which allows us to predict *exactly* the work performed by each node in *any* distributed execution. A node in layer j will reverse exactly j times before stabilization. Our lower and upper bounds follow easily from the exact analysis.

Partial reversal algorithm. For the partial reversal algorithm, we show that when started from an initial state with n bad nodes, the work and time needed to stabilize is $O(n \cdot a^* + n^2)$, where a^* corresponds to the difference between the maximum and minimum heights of the nodes in the initial state. This bound is tight. We show that there are networks with initial states which require $\Omega(n \cdot a^* + n^2)$ time for stabilization.

The a^* value can grow unbounded as topological changes occur in the network. Consequently, in the worst case, the full reversal algorithm outperforms the partial reversal algorithm.

Deterministic algorithms. We show a lower bound on the worst-case work and time until stabilization for *any* deterministic reversal algorithm. We show that for any deterministic reversal algorithm on a given graph, there exists an initial state such that if a bad node d hops away from its closest good node, then it has to reverse d times before stabilization. Using this, we further show that there exist networks and initial states with n bad nodes such that the algorithm needs $\Omega(n^2)$ work and time until stabilization. As a consequence, from the worst-case perspective, the full reversal algorithm is work and time optimal, while the partial reversal algorithm is not.

Equivalence of executions. We show that for any deterministic reversal algorithm, all distributed executions of the algorithm starting from the same initial state are

equivalent: (1) the resulting final state of the network upon stabilization is the same, and (2) each node performs the same number of reversals until stabilization in all executions. As a result, the work of the algorithm is independent of the execution schedule.

1.3. Related work. Link reversal algorithms were introduced by Gafni and Bertsekas in [7], where they provide a proof that shows that a general class of link reversal algorithms, including the partial and full reversal algorithms, eventually stabilize when started from any initial state. However, they do not give work and time bounds.

The TORA [10] builds on a variation of the GB partial reversal algorithm and adds a mechanism for detecting and dealing with partitions (disconnected components) in the network. The practical performance of the TORA has been studied in [11]. A variant of a link reversal routing algorithm is the lightweight mobile routing (LMR) algorithm [5, 6]. An overview of link reversal routing algorithms can be found in [12, Chapter 8]. A performance comparison of various ad hoc routing algorithms, including TORA, is presented in [1]. Further surveys can be found in [13, 14].

Malpani, Welch, and Vaida [9] build a mobility aware leader election algorithm on top of TORA and present partial correctness proofs (TORA does not have any) showing the stability of the algorithm. Although all the above work use link reversal, none of them have any formal performance analysis.

In the context of distributed sensor networks, coordination algorithms which are based on the paradigm of *directed diffusion* [8] are closely related to link reversal algorithms. For example, Intanagonwiwat et al. [8] state that their algorithm is closest to the TORA algorithm [10] in its attempt to localize the repairs due to node failures. Hence, our analysis also might lead to a better understanding of the performance of directed diffusion.

Link reversal algorithms attempt to always maintain routes to destinations in ad hoc networks. In cases when the network is sparsely populated with nodes, or when the rate of topology changes is too high, it may be infeasible to maintain such paths to the destination. In such cases, other strategies are needed for data delivery, such as those in [3, 4] which do not maintain paths to destination at all, but instead transmit data through strategies based on gossiping.

Outline of the paper. The rest of the paper is organized as follows. Section 2 contains a description of the GB partial and full reversal algorithms as well as a definition of deterministic algorithms. In section 3 we show the equivalence of executions of a given deterministic algorithm. Sections 4 and 5 contain the analyses of the full and partial reversal algorithms, respectively. In section 6, we show the general lower bound for deterministic link reversal algorithms, and we conclude with a discussion and open problems in section 7.

2. Link reversal algorithms. We assume that each node has a unique integer id and denote the node with id i by v_i . The nodes have heights which are guaranteed to be unique (ties broken by node ids) and are chosen from a totally ordered set. A link is always directed from the node of greater height to the node of the smaller height. The destination has the smallest height, and it is a special kind of sink which never reverses. Since any directed path in such a graph always proceeds in the direction of decreasing height, *the directed graph will always be a directed acyclic graph (DAG)*. This is a significant feature, since the algorithms need not make further effort to maintain acyclicity in routing, and the graph remains acyclic even if topological changes occur.

If the underlying graph is connected, the link reversal algorithms bring the directed graph from its initial state to a state in which it is destination oriented. In our analysis, we consider only connected graphs. Note that there could possibly be multiple paths from any node to the destination. We now describe the two GB algorithms, adapting the discussion from [7], and then we define the class of deterministic algorithms.

Full reversal algorithm. In the full reversal algorithm, when a node becomes a sink it simply reverses the directions of all its incoming links (see top part of Figure 1, which is adapted from [7]). The algorithm can be implemented with heights as follows. The height h_i of node v_i is the pair (a_i, i) (the second field is used to break ties). The height of the destination (say v_d) is $(0, d)$. Heights are ordered lexicographically. If v_i is a sink, then its height upon reversal is updated to be larger than the heights of all its neighbors. Let $N(v_i)$ denote the set of adjacent nodes to v_i . Formally, the height of v_i after its reversal is $(\max\{a_j \mid v_j \in N(v_i)\} + 1, i)$.

Partial reversal algorithm. In the partial reversal algorithm, the height of each node v_i is a triple (a_i, b_i, i) . As in full reversal, node v_i reverses only when it becomes a sink. The height of v_i after reversal is greater than the height of at least one neighbor, but may not be greater than the height of every neighbor. The height of the destination v_d is $(0, 0, d)$. Heights are ordered lexicographically. The second field b_i helps the sink avoid reversing links toward adjacent nodes, which have caused the node to become a sink in the first place. Thus, reversals are not immediately propagated to parts of the network which have already reversed. Formally, let $\bar{h}_i = (\bar{a}_i, \bar{b}_i, i)$ denote the height of v_i after its reversal. See the bottom part of Figure 1 (adapted from [7]) for an example execution of the partial reversal algorithm. The partial reversal algorithm updates heights as follows:

- $\bar{a}_i = \min\{a_j \mid v_j \in N(v_i)\} + 1$.
- $\bar{b}_i = \min\{b_j \mid v_j \in N(v_i) \text{ and } \bar{a}_i = a_j\} - 1$ if there exists a neighbor v_j with $\bar{a}_i = a_j$; otherwise, $\bar{b}_i = b_i$.

The basic idea behind these functions is as follows. In a network state I , where v_i is a sink, we can divide the neighbors of v_i into two categories: (i) $\{v_j \mid a_j = a_i\}$ and (ii) $\{v_j \mid a_j > a_i\}$. Node v_i must have reversed after the last reversal of every node in category (i) since, otherwise, those nodes would have $a_j \geq a_i + 1$. On the other hand, nodes of category (ii) must have reversed after the last reversal of node v_i since, otherwise, the heights of those nodes would not be higher than a_i . Therefore, node v_i is a sink in state I due only to nodes of category (ii). Thus, when v_i reverses after state I , its new height should be set so that the links point only toward nodes of category (i). This is achieved by setting $\bar{a}_i = a_i + 1$. In order to make the nodes of category (ii) to point to v_i , we need only take care of nodes with $a_j = \bar{a}_i$, for which we adjust \bar{b}_i to be lower than all the corresponding b_j 's.²

Deterministic algorithms. A deterministic reversal algorithm is defined by a ‘‘height increase’’ function g . We assume that the heights are chosen from some totally ordered universe and that the heights of different nodes are unique. If node v is a sink of degree k , whose current height is h_v , and adjacent nodes v_1, v_2, \dots, v_k have heights h_1, h_2, \dots, h_k , respectively, then v 's height after reversal is $g(h_1, h_2, \dots, h_k, h_v)$. Function g is such that the sink reverses at least one of its incoming links. The GB full and partial reversal algorithms are deterministic.

²The function for the update of \bar{a}_i is expressed in terms of the minimum value of the neighbors, since topological changes might generate a state in which no neighbor has $a_j = a_i$.

3. Equivalence of executions. In this section, we prove some properties about deterministic link reversal algorithms. The main result of this section is that for any deterministic reversal algorithm, all executions that start from the same initial state are essentially equivalent: the resulting final state of the network upon stabilization is the same. We actually show the stronger result that each node performs the same number of reversals, with the same heights, until stabilization in all executions. We first give some basic definitions for states and executions; then we define the dependency graph, which will help to show that all executions are equivalent, and finally, we give the main result.

3.1. States, executions, and dependency graphs. Here we give basic definitions for states and executions. At any configuration of the network, the *node state* of a node v is defined as the current height of v . The *network state* is defined as the collection of the individual states of all the nodes in the network. Note that the network state uniquely determines the directions of all the links in the network.

A node reversal r is defined as a tuple $r = (v, h, H)$, where v is the sink executing the reversal, h is v 's height before reversal, and H is the set of the heights of all of v 's neighbors before the reversal. Given an initial state containing bad nodes, an *execution* E is defined as a sequence of reversals $E = r_1, r_2, \dots, r_k$, where $r_i = (v_i, h_i, H_i)$, and $1 \leq i \leq k$. A *complete execution* is defined as an execution that ends in a destination oriented graph; unless otherwise stated, we will refer to complete executions from here on.

Clearly, there are many possible executions starting from the same initial state. We give the following definition for equivalent executions.

DEFINITION 3.1. *Starting from the same initial state, two executions are equivalent if they give the same final state.*

In order to show that two executions are equivalent, we will use the dependency graphs of the executions which we define next. Any execution imposes a partial order on the reversals. The partial order induced by execution $E = r_1, r_2, \dots, r_k$ is defined as a directed graph whose nodes are the reversals $r_i, i = 1, \dots, k$. There is a directed edge from $r_i = (v_i, h_i, H_i)$ to $r_j = (v_j, h_j, H_j)$ if

- v_j is a neighbor of v_i , and
- r_j is the first reversal of v_j after r_i in execution E .

We will refer to this graph as the *dependency graph* of execution E . Intuitively, if there is a directed path between reversals r_i and r_j in the dependency graph, then the order of these two reversals cannot be interchanged. Moreover, if there is no directed path from r_i to r_j , then these two reversals are independent and can be performed in parallel (in the same time step).

We define the *depth* of a reversal in the dependency graph as follows. A reversal that does not have any incoming edges has depth 1 (these are the reversals of the nodes which are sinks in the initial state). The depth of any other reversal r is one more than the maximum depth of a reversal which points to r . The depth of the dependency graph is the maximum depth d of any reversal in the graph. The dependency graph is important for the following reason.

Fact 1. The dependency graph of an execution uniquely determines

- the final state of the network,
- the number of reversals performed by each node, and
- the stabilization time when all sinks reverse simultaneously, which is the depth of the dependency graph d .

3.2. Proof of equivalence. We show that all executions of a link reversal algorithm give the same dependency graph, which implies that the executions are equivalent. Fact 1 implies that this result is actually stronger than simply showing that executions are equivalent. We first show two lemmas that will be of use in further proofs.

LEMMA 3.2. *For any reversal algorithm starting from any initial state, a good node never reverses until stabilization. Further, a good node always remains good until stabilization.*

Proof. If v is a good node, then by definition there exists a path $v = v_k, v_{k-1}, \dots, v_1, v_0 = s$, where s is the destination, and there is an edge directed from v_i to v_{i-1} for each $i = 1, \dots, k$.

For each $i = 0, \dots, k$, we prove that node v_i never reverses, using induction on i . The base case ($i = 0$) is obvious since the destination never reverses. Suppose the hypothesis is true for $i = l < k$. Then v_l never reverses, so that the edge between v_{l+1} and v_l is always directed from v_{l+1} to v_l . Thus, there is always an outgoing edge from v_{l+1} , which implies that v_{l+1} never reverses, and completes the proof by induction.

This also implies that the directed path $v = v_k, v_{k-1}, \dots, v_1, v_0 = s$ always exists in the network, showing that node v remains good. \square

LEMMA 3.3. *If a node v is a sink, then v remains a sink until it reverses. Further, v eventually reverses.*

Proof. If a node v is a sink, then clearly none of its neighbors can be sinks at the same time, and hence they cannot reverse. Thus, the only node that can change the direction of the incoming links to v is v itself. Reversals by other nodes in the network do not affect this. Thus, v remains a sink until it reverses.

Further, the reversal of v is enabled continuously until v actually reverses. Since we assume that the distributed system eventually makes progress (an action that is continuously enabled will eventually take place), v eventually reverses. \square

THEOREM 3.4 (identical dependency graphs). *All executions of a deterministic reversal algorithm starting from the same initial state give identical dependency graphs.*

Proof. Consider two executions of the algorithm starting from the same initial state, say, execution $R = r_1, r_2, \dots$ and execution $S = s_1, s_2, \dots$. Let p_R and p_S be the dependency graphs induced by R and S , respectively. We will show that p_R and p_S are identical.

We will show by induction that, for every $k = 1, 2, \dots$, the induced subgraph of p_R consisting of vertices at depths k or less is identical to the similarly induced subgraph of p_S consisting of vertices at depths of k or less.

Base case $k = 1$. Consider any reversal $r = (v, h, H)$ in p_R at depth 1. Since r does not have any incoming edges in p_R , node v must be a sink in the initial state of the network. From Lemma 3.3, v must also reverse in S . Since h and H are the heights of v and its neighbors, respectively, in the initial state, and they do not change until v reverses at least once, the first reversal of v in S is also (v, h, H) , and is at depth 1. Similarly, any other reversal at depth 1 in p_S is also a reversal at depth 1 in p_R , and this proves the base case.

Inductive case. Suppose the hypothesis is true for all $k < l$. We show that it is also true for $k = l$. Consider any reversal $r = (v, h, H)$ at depth l in p_R . We show that this reversal is also present in p_S with the same set of incoming edges. Let U be the set of reversals that are pointing into r in p_R . Once all reversals in U are executed, node v becomes a sink in execution R . From the inductive step, all reversals in U are

also present in p_S , and hence in S . We examine two cases.

Case 1. Reversal r is the first reversal of v in R . Then, the execution of all reversals in U will also cause v to be a sink in S . Thus v also will reverse in S . Its height before reversal in S is h , since the height has not changed from the initial state. Consider the heights of v 's neighbors before v 's reversal in S . These are equal to H . The reason is as follows. The neighbors of v who haven't reversed so far in S have the same height as in the initial state. The other neighbors are present in U , and hence their heights are the same as in H . Thus, there is a reversal (v, h, H) at depth l in p_S whose incoming edges are the same as in p_R .

Case 2. Reversal r is not the first reversal of v in R . Let r' denote the previous reversal of v in R . Since r' is at a lower depth in p_R than r , by the induction hypothesis, r' is also present in p_S . After reversal r' , node v will be in the same state in both R and S . After the reversals in U , v 's neighbors will be in the same state in S as in R . Thus, the reversal (v, h, H) is also present in S at depth l with the same incoming edges as in p_R .

Thus, we have shown that every node at depth l in p_R is present at depth l of p_S , with the same incoming edges. The same argument goes the other way too: every node in p_S is present in p_R . This proves the inductive case for $k = l$, and concludes the proof. \square

The following corollary follows from Fact 1 and Theorem 3.4.

COROLLARY 3.5 (equivalence of executions). *All executions of a deterministic reversal algorithm starting from the same initial state are equivalent. Moreover,*

- *the number of reversals of each node in every execution is the same, and*
- *when all sinks reverse simultaneously, the stabilization time of every execution is d , the depth of the (unique) dependency graph.*

4. Full reversal algorithm. In this section, we present the analysis of the full reversal algorithm. We present a decomposition of the bad nodes in the initial network state into *layers*, which allows us to predict *exactly* the work performed by each node in any distributed execution until stabilization: a node at layer i will reverse exactly i times. From the exact analysis, we obtain worst-case bounds for the work and time needed for stabilization.

4.1. State sequence for full reversal. In order to obtain the exact analysis, we first show that, starting from any initial state, there exists an execution which consists of consecutive execution segments such that at each execution segment, each remaining bad node reverses exactly once. We will then use this result to determine the exact number of reversals of each bad node in the layer decomposition.

In particular, consider some initial state I_1 of the graph which contains bad nodes. We will show that there is an execution $E = E_1, E_2, E_3, \dots$, and states I_1, I_2, I_3, \dots , such that execution segment E_i , $i \geq 1$, brings the network from a state I_i to a state I_{i+1} , and in E_i each bad node of I_i reverses exactly one time. In order to show that E exists, we need to prove the following two lemmas.

LEMMA 4.1. *Consider a state I in which a node v is bad. Then, node v will reverse at least one time before it becomes a good node.*

Proof. If v is a sink, then clearly node v has to reverse at least one time. Now consider the case when v is not a sink in state I . Suppose, for contradiction, that node v becomes good without performing any reversals after state I . Consider an execution which brings the graph from state I to a state I^g in which node v is good. A *nonreversed* node is any node w such that in state I node w is bad, while in state I^g node w is good, and w does not reverse between I and I^g . Since in state I^g node

v is good, there must exist in I^g a directed path $v, v_1, \dots, v_{k-1}, v_k$, $k \geq 1$, in which v_k is good in I^g and I .

We will show that nodes v_1, \dots, v_{k-1} are nonreversed. Consider node v_1 . Assume for contradiction that node v_1 has reversed between states I and I^g . Since in I^g there is a link directed from node v to node v_1 , and v_1 has reversed between states I and I^g , it must be that node v has reversed at least one time, a contradiction. Thus, node v_1 is nonreversed. Similarly, using induction, we can easily show that nodes v_2, \dots, v_{k-1} are also nonreversed. Since nodes v_1, \dots, v_{k-1} are nonreversed, it has to be that in state I there is a directed path $v, v_1, \dots, v_{k-1}, v_k$. Thus, in state I node v is a good node. This contradiction completes the proof. \square

LEMMA 4.2. *Consider some state I which contains bad nodes. There exists an execution which brings the network from state I to a state I' (not necessarily a final state) such that every bad node of state I reverses exactly one time.*

Proof. Suppose for contradiction that there is no such execution. Then, there exists an execution E^f which brings the system from state I to a state I^f such that the following conditions hold:

1. There is at least one bad node in I which hasn't reversed in E^f . Let A denote the set of such bad nodes of I .
2. Any other bad node v of I , with $v \notin A$, has reversed exactly one time. Let B denote the set of such bad nodes of I .
3. The number of nodes in set B is maximal.

First we show that all the nodes that are sinks in state I^f have to be members of set B . Suppose that a sink in state I^f is a member of set A . Then the sink hasn't reversed since state I . If the sink reverses then it could be an additional member of set B . Thus, B is not maximal as required by the third condition. Therefore, the sink has to be a member of B .

Next we show that at least one node in A is a sink in state I^f , which proves that execution E^f does not exist. Assume for contradiction that no node of A is a sink in I^f . Then, each node in A has an outgoing edge in I^f . These outgoing edges from A cannot point toward nodes in B , since the nodes in B have reversed their edges, while the nodes in A haven't. Moreover, these outgoing edges cannot point toward good nodes of state I , since this would imply that nodes in A are good in state I^f , while Lemma 4.1 implies that each node of set A remains bad in state I^f . Thus, these outgoing edges must point toward nodes in set A . Since each node in set A has an outgoing edge in set A , it must be, from the pigeonhole principle, that there is a walk in which a node in A is repeated. Thus, there is a cycle in the graph, violating the fact that the graph is acyclic. Thus, it must be that a node in A is a sink, a contradiction. \square

Lemma 4.2 implies that the execution segments E_i and the states I_i exist. The *link-state* of a node v is the vector of directions of its incident links. We show that each execution segment leaves the link-state of bad nodes unchanged for the bad nodes, which are not adjacent to good nodes.

LEMMA 4.3. *If in state I_i , $i \geq 1$, node v is bad and v is not adjacent to a good node, then v will remain in the same link-state in I_{i+1} .*

Proof. Let $A(v)$ denote the set of nodes adjacent to v in state I_i . Since all nodes in $A(v)$ are bad in state I_i , each of them reverses in execution E_i . Moreover, v also reverses in E_i . These reversals leave the directions of the links between v and $A(v)$ in state I_{i+1} the same as in state I_i . \square

4.2. Layers for full reversal. Here, we show that given some initial state I with bad nodes, it is possible to decompose the bad nodes into layers and determine the exact number of reversals for the nodes of each layer until stabilization: a node in layer i reverses exactly i times.

In particular, we decompose the bad nodes into layers $L_1^I, L_2^I, \dots, L_m^I$, defined inductively as follows (see Figure 2). A bad node v is in layer L_1^I if the following conditions hold:

- There is an incoming link to node v from a good node, or
- there is an outgoing link from node v to a node in layer L_1^I .

A node v is in layer L_k^I , $k > 1$, if k is the smallest integer for which one of the following hold:

- There is an incoming link to node v from a node in layer L_{k-1}^I , or
- there is an outgoing link from node v to a node in layer L_k^I .

From the above definition, it is easy to see that any node of layer L_k^I , where $k > 1$, can be connected only with nodes in layers L_{k-1}^I , L_k^I , and L_{k+1}^I . The nodes of layer L_1^I are the only ones that can be connected with good nodes. The links connecting two consecutive layers L_{k-1}^I and L_k^I can be directed only from L_{k-1}^I to L_k^I . Note that the number of layers m is not greater than the number of bad nodes in the network n .

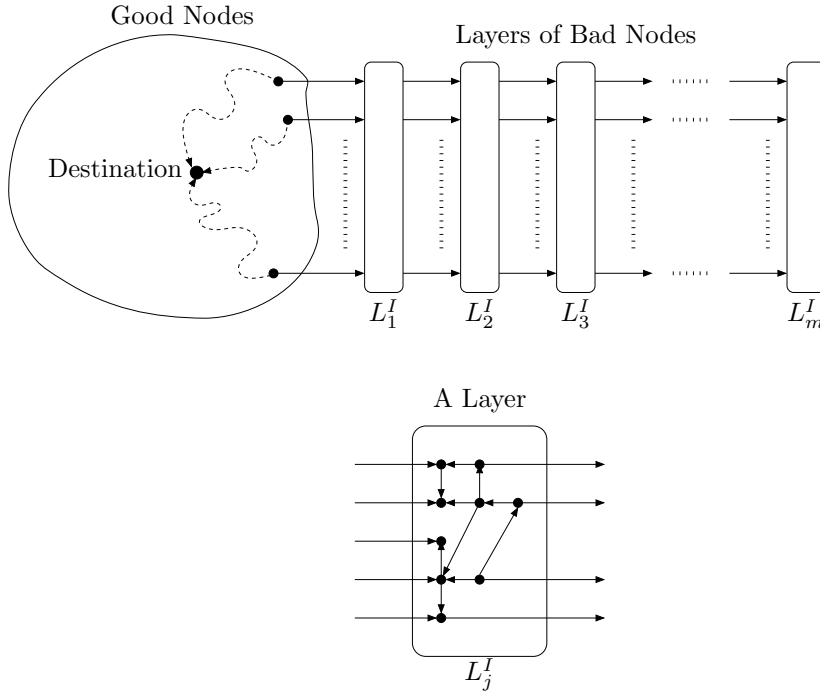


FIG. 2. Decomposition of the bad nodes into layers.

Consider now the states I_1, I_2, \dots and execution segments E_1, E_2, \dots , as described in section 4.1. For each of these states we can divide the bad nodes into layers, as described above. In the following sequence of lemmas we will show that the layers of state I_1 become good one by one at the end of each execution segment E_i , $i \geq 1$. We show now that the first layer of state I_i becomes good at the end of execution E_i .

LEMMA 4.4. *At the end of execution E_i , $i \geq 1$, all the bad nodes of layer $L_1^{I_i}$ become good, while all the bad nodes in layers $L_j^{I_i}$, $j > 1$, remain bad.*

Proof. First we show that the bad nodes of layer $L_1^{I_i}$ become good. There are two kinds of bad nodes in layer $L_1^{I_i}$ at state I_i : type α , nodes which are connected with an incoming link to a good node; and type β , nodes which are connected with an outgoing link to another node in layer $L_1^{I_i}$.

It is easy to see that there is a direct path from any β node to some α node, consisting of nodes of layer $L_1^{I_i}$. Since all bad nodes reverse exactly once in execution E_i , all α nodes become good in state I_{i+1} . Moreover, from Lemma 4.3, the paths from β nodes to α remain the same in state I_{i+1} . Thus, the β nodes also become good in state I_{i+1} . Therefore, all the bad nodes of layer $L_1^{I_i}$ become good in state I_{i+1} .

Now we show that the bad nodes in layers $L_j^{I_i}$, $j > 1$, remain bad in state I_{i+1} . From Lemma 4.3, in state I_{i+1} , the links connecting layers $L_1^{I_i}$ and $L_2^{I_i}$ are directed from $L_1^{I_i}$ to $L_2^{I_i}$. Thus, in state I_{i+1} , there is no path connecting nodes of layer $L_2^{I_i}$ to good nodes. Similarly, there is no path from the nodes of layer $L_j^{I_i}$, for any $j > 2$, to good nodes. Thus all nodes in layers $L_j^{I_i}$, $j > 1$, remain bad. \square

We now show that the basic structure of layers of the bad nodes remains the same from state I_i to state I_{i+1} , with the only difference being that the first layer of I_{i+1} is now the second layer of I_i .

LEMMA 4.5. $L_j^{I_{i+1}} = L_{j+1}^{I_i}$, $i, j \geq 1$.

Proof. From Lemma 4.4, at the end of execution E_i , all the bad nodes of layer $L_1^{I_i}$ become good, while all the bad nodes in layers $L_j^{I_i}$, $j > 1$, remain bad. From Lemma 4.3 all bad nodes in layers $L_j^{I_i}$, $j > 1$, remain in the same link-state in I_{i+1} as in I_i . Therefore, $L_j^{I_{i+1}} = L_{j+1}^{I_i}$, $j \geq 1$. \square

From Lemmas 4.4 and 4.5, we have that the number of layers is reduced by one from state I_i to state I_{i+1} . If we consider the layers of the initial state I_1 , we have that all the bad nodes in the layers become good one by one at the end of executions E_1, E_2, E_3, \dots in the order $L_1^{I_1}, L_2^{I_1}, L_3^{I_1}, \dots$. Since in each execution E_i all the bad nodes reverse exactly one time, we obtain the following.

LEMMA 4.6. *Each node in layer $L_j^{I_1}$, $j \geq 1$, reverses exactly j times before it becomes a good node.*

From Corollary 3.5, we know that all possible executions when started from the same initial state require the same number of reversals. Thus, the result of Lemma 4.6, which is specific to the particular execution E , applies to all possible executions. Therefore, we obtain the following theorem.

THEOREM 4.7 (exact number of reversals for full reversal). *For any initial state I , and any execution of the full reversal algorithm, L_1^I, L_2^I, \dots is a division of the bad nodes in I into layers such that each node in layer L_j^I , $j \geq 1$, reverses exactly j times before it becomes a good node.*

4.3. Worst-case bounds for full reversal. We now give worst-case upper and lower bounds for the work and time needed for stabilization by the full reversal algorithm. Both bounds are obtained with the use of Theorem 4.7.

From Theorem 4.7, we have that for any initial state I , each node in layer L_j^I reverses exactly j times until it becomes good. Thus, the total number of reversals of the nodes of layer j is $j \cdot |L_j^I|$. If there are m layers of bad nodes, the total number of reversals is $\sum_{j=1}^m j \cdot |L_j^I|$. If I has n bad nodes, there are at most n layers in the worst case (each layer contains one bad node). Thus, each node reverses at most n times. Since there are n bad nodes, the total number of reversals in the worst case

is $O(n^2)$. Moreover, since a node reversal takes one time step and in the worst case all reversals are executed sequentially, the total number of reversals gives an upper bound on the stabilization time. Thus, we have the following.

COROLLARY 4.8 (work and time upper bounds for full reversal). *For any graph with an initial state with n bad nodes, the full reversal algorithm requires at most $O(n^2)$ work and time until stabilization.*

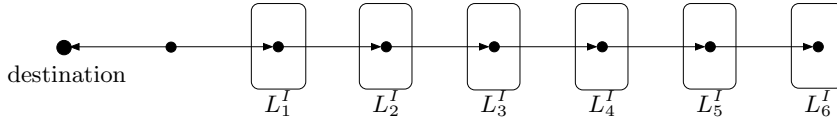


FIG. 3. Worst-case work for full reversal: graph G_1 with $n = 6$ bad nodes.

Actually, the upper bound of Corollary 4.8 is tight in both work and time in the worst case. First we show that the work bound is tight. Consider a graph G_1 which is an initial state with n layers of bad nodes such that each layer has exactly one node (see Figure 3 with $n = 6$). From Theorem 4.7, each node in the i th layer will reverse exactly i times. Thus, the sum of all the reversals performed by all the bad nodes is $n(n + 1)/2$, leading to the following corollary.

COROLLARY 4.9 (work lower bound for full reversal). *There is a graph with an initial state containing n bad nodes such that the full reversal algorithm requires $\Omega(n^2)$ work until stabilization.*

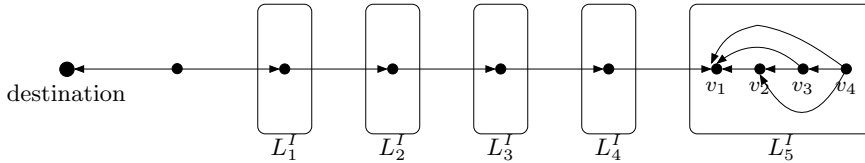


FIG. 4. Worst-case stabilization time for full reversal: graph G_2 with $n = 8$ bad nodes, $m_1 = 5$ layers, and $m_2 = 4$ nodes in layer m_1 .

We will now show that the time bound of Corollary 4.8 is tight (within constant factors) in the worst case. Consider a graph G_2 (see Figure 4) with an initial state in which there are n bad nodes, such that it consists of $m_1 = \lfloor n/2 \rfloor + 1$ layers. The first $m_1 - 1$ layers contain one node each, while the last layer contains $m_2 = \lceil n/2 \rceil$ nodes. The last layer m_1 is as follows: there are m_2 nodes v_1, v_2, \dots, v_{m_2} . Node v_i has outgoing links to all nodes v_j such that $j < i$. The node of layer $m_1 - 1$ has an outgoing link to node v_1 (see Figure 4).

From Theorem 4.7, we know that each node in layer m_1 requires exactly m_1 reversals before it becomes good. Since there are m_2 nodes in layer m_1 , $m_1 \cdot m_2 = \Omega(n^2)$, reversals are required before these nodes become good. The key point is that any two nodes in layer m_1 are adjacent, so that all the reversals in that layer have to be performed sequentially. Thus, the reversals in layer m_1 alone take $\Omega(n^2)$ time, providing the following corollary.

COROLLARY 4.10 (time lower bound for full reversal). *There is a graph with an initial state containing n bad nodes such that the full reversal algorithm requires $\Omega(n^2)$ time until stabilization.*

Note that Corollary 4.10 subsumes Corollary 4.9, since a lower bound on time is also a lower bound on work.

5. Partial reversal algorithm. In this section, we present the analysis of the partial reversal algorithm. We first give an upper bound for work and stabilization time. We then present lower bounds for a class of worst-case graphs which is used to show that the upper bound is tight.

5.1. Upper bounds for partial reversal. Given an arbitrary initial state I , we give an upper bound on the work and stabilization time needed for the partial reversal algorithm. In order to obtain the bound, we decompose the bad nodes into levels and give an upper bound for the number of reversals of the nodes in each level; this then gives us an upper bound on work and time.

In particular, suppose that initial state I of the network contains n bad nodes. We say that a bad node v of state I is in *level* i if the shortest undirected path from v to a good node has length i . Note that the number of levels is no more than n .

The upper bound depends on the minimum and maximum heights of the nodes in state I . According to the partial reversal algorithm, each node v_i has a height (a_i, b_i, i) . We will refer to a_i as the *alpha value* of node v_i . Let a^{\max} and a^{\min} denote the respective maximum and minimum alpha values of any node in the network in state I . Let $a^* = a^{\max} - a^{\min}$. We first give an upper bound on the alpha value of any node upon stabilization.

LEMMA 5.1. *After a node in level i becomes good its alpha value never exceeds $a^{\max} + i$.*

Proof. We prove the claim by induction on the number of levels. For the induction basis, consider a node v in level 1. If the alpha value of v becomes at least $a^{\max} + 1$, then v must have become a good node, since its height is more than the height of at least one adjacent node v' which is good in state I (from Lemma 3.2 v' does not reverse, and thus its alpha value remains at most a^{\max}). We need only show that during its final reversal, the alpha value of v will not exceed $a^{\max} + 1$. According to the partial reversal algorithm, the alpha value of v is equal to the smallest alpha value of its neighbors plus one. Moreover, the smallest alpha value of the neighbors cannot be greater than a^{\max} , since in I node v is adjacent to good nodes which don't reverse in future states (a consequence of Lemma 3.2). Thus, the alpha value of v will not exceed $a^{\max} + 1$ when v becomes a good node. Further, from Lemma 3.2, the alpha value of node v will not change thereafter.

For the induction hypothesis, let's assume that the alpha value of any node in level i , where $1 \leq i < k$, does not exceed $a^{\max} + i$, after that node becomes good. For the induction step, consider layer L_k . Let v be a node in level k . Clearly, node v is adjacent to some node in level $k - 1$. From the induction hypothesis, the alpha value of every node in level $k - 1$ cannot exceed $a^{\max} + (k - 1)$ in any future state from I . If the alpha value of v becomes at least $a^{\max} + k$, then v must have become a good node, since its height is more than that of the adjacent nodes in level $k - 1$ when these nodes become good. We need only show that during its final reversal, the alpha value of v will not exceed $a^{\max} + k$. According to the partial reversal algorithm, the alpha value of v is not more than the smallest alpha value of its neighbors plus one. Moreover, the smallest alpha value of the neighbors cannot exceed $a^{\max} + (k - 1)$, which is the maximum alpha value of the nodes in level $k - 1$ when these nodes become good. Thus, the alpha value of v will not exceed $a^{\max} + k$ when v becomes a good node. Further, from Lemma 3.2, the alpha value of node v will not change thereafter. \square

At each reversal, the alpha value of a node increases by at least 1. Since the alpha value of a node can be as low as a^{\min} , Lemma 5.1 implies that a node in level i reverses at most $a^{\max} - a^{\min} + i$ times. Furthermore, since there are at most n levels, we obtain the following corollary.

COROLLARY 5.2. *A bad node will reverse at most $a^* + n$ times before it becomes a good node.*

Considering now all the n bad nodes together, Corollary 5.2 implies that the work needed until the network stabilizes is at most $n \cdot a^* + n^2$. Since in the worst case the reversal of the nodes may be sequential, the upper bound for work is also an upper bound for the time needed to stabilize. Thus we have the following.

THEOREM 5.3 (work and time upper bounds for partial reversal). *For any initial state with n bad nodes, the partial reversal algorithm requires at most $O(n \cdot a^* + n^2)$ work and time until the network stabilizes.*

5.2. Lower bounds for number of reversals. We will show that the upper bounds on work and time given in Theorem 5.3 are tight. We construct a class of worst-case graphs with initial states which require as much work and time as the upper bounds. In order to prove the lower bounds, we first determine how many reversals each node performs in the network.

In particular, consider a graph with an initial state I containing n bad nodes which can be decomposed into an even number m of layers $L_1, L_2, \dots, L_{m-1}, L_m$ in the following way. A node is a *source* if all the links incident to the node are outgoing. The odd layers L_1, L_3, \dots, L_{m-1} contain only nodes which are nonsources, while the even layers L_2, L_4, \dots, L_m contain only nodes which are sources. The nodes in layer L_1 are the only bad nodes adjacent to good nodes. Let G denote the set of good nodes adjacent to layer L_1 . Nodes in layer L_i may be adjacent only to nodes of the same layer and layers L_{i-1} and L_{i+1} .³ We actually require that each node of L_i is adjacent to at least one node of L_{i-1} and at least one node of L_{i+1} . In addition, state I is taken so that all good nodes in the network have alpha value a^{\max} , while all the bad nodes have alpha value a^{\min} , where $a^{\max} > a^{\min}$. Let $a^* = a^{\max} - a^{\min}$. Instances of such an initial state are shown in Figures 5 and 6; at the end of this section we describe how to obtain such configurations with arbitrary large a^{\max} in a mobile ad hoc network.

Given such an initial state I , we will give a lower bound on the number of reversals performed by each node at each layer until the network stabilizes. In order to obtain this result, we first show some necessary lemmas. A *full reversal* is a reversal in which a node reverses all of its links. Note that after a full reversal, a node becomes a source. We show that bad nodes which are sources always perform full reversals whenever they become sinks.

LEMMA 5.4. *Consider any state I_1 of the network in which a bad node v is a source with alpha value a . In a subsequent state I_2 , in which node v becomes a sink for the first time after state I_1 , the following occur: (1) v performs a full reversal, and (2) after the reversal of v , the alpha value of v becomes $a + 2$.*

Proof. In state I_1 , since v is a source, all the adjacent nodes of v have alpha value at most a . Between states I_1 and I_2 , each adjacent node of v has reversed at least once. We will show that in state I_2 , the alpha value of each adjacent node of v is $a + 1$.

Let w be any adjacent node of v . First, we show that the alpha value of w in I_2 is at least $a + 1$. If in I_2 the alpha value of w is less than a , then v must have an outgoing link toward w , and thus v cannot possibly be a sink in I_2 , a contradiction. Therefore, in I_2 the alpha value of w has to be at least a . Next, we show that this alpha value cannot be equal to a . If the alpha value of w in I_2 is a , then it must

³If $i = 1$, substitute G for L_{i-1} . If $i = m$, don't consider L_{i+1} .

be that the alpha value of w in I_1 was less than a (since w reversed between I_1 and I_2 and points toward v). When w was a sink the last time before I_2 , w must have been adjacent to another node u with height $a - 1$. When w reversed, its alpha value became a , but its incoming link from v didn't change direction since u had a smaller alpha value. Thus v cannot possibly be a sink in I_2 , a contradiction. Therefore, the alpha value of w in I_2 cannot be equal to a , and it has to be at least $a + 1$.

Next, we show that the alpha value of w cannot be greater than $a + 1$. When w reverses, its alpha value is at most the minimum alpha value of its neighbors plus one. Therefore, since v is a neighbor of w with alpha value a , when w reverses, its alpha value cannot exceed $a + 1$.

Therefore, the alpha value of w in state I_2 is exactly $a + 1$. This implies that in I_2 all the neighbors of v have alpha value $a + 1$. Thus, when v reverses, it performs a full reversal and its alpha value becomes $a + 2$. \square

Given state I described above, we give a lower bound for the alpha values of the nodes in each layer when the network stabilizes.

LEMMA 5.5. *When the network stabilizes from state I , the alpha values of all the nodes in layers L_{2i-1} and L_{2i} , $1 \leq i \leq m/2$, are at least $a^{\max} + i$.*

Proof. Let I' denote the state of the network when it stabilizes. We prove the claim by induction on i . For the basis case, where $i = 1$, we consider layers L_1 and L_2 . In state I , all the nodes of layer L_1 have only incoming links from G . In state I' , there must exist a set S , consisting of nodes from L_1 , such that the nodes in S have outgoing links pointing toward G .

Let v be a node in S . In state I' , the alpha value of v is at least a^{\max} , since the nodes in G have alpha value a^{\max} . Actually, we will show that the alpha value of v in I' is larger than a^{\max} . Assume for contradiction that this value is a^{\max} . When node v reversed and obtained the alpha value a^{\max} , it cannot possibly have reversed its links toward G since, for these links, v adjusted only its second field on its height. Thus, in state I' node v is still bad, a contradiction. Therefore, in state I' , node v has alpha value at least $a^{\max} + 1$; thus, in state I' , all nodes in set S have alpha value at least $a^{\max} + 1$.

Now, consider the rest of the nodes in layers L_j , $j \geq 1$. Let w be any such node. In state I' , w is good, and thus there exists a directed path from w to a good node in G . This path has to go through the nodes of S ; thus each node in the path must have alpha value at least $a^{\max} + 1$, which implies that w has alpha value at least $a^{\max} + 1$. Therefore, in state I' , all nodes in L_1 and L_2 (including S) have alpha value at least $a^{\max} + 1$.

Now, let's assume that the claim holds for all $1 \leq i < k$. We will show that the claim is true for $i = k$. We consider layers L_{2k-1} and L_{2k} . In state I all the nodes of layer L_{2k-1} have only incoming links from L_{2k-2} . In state I' , there must exist a set S , consisting of nodes of L_{2k-1} , such that the nodes in S have outgoing links pointing toward L_{2k-2} . The rest of the proof is similar to the induction basis, where now we show that the nodes in S in state I' , have alpha values at least $a^{\max} + k$, which implies that all nodes in L_{2k-1} and L_{2k} have alpha value at least $a^{\max} + k$. \square

We are now ready to show a central theorem for the lower bound analysis, which is a lower bound on the number of reversals for the nodes of each layer. This result will help us to obtain lower bounds for work and time needed for stabilization.

THEOREM 5.6 (lower bound on reversals for partial reversal). *Until the network stabilizes, each node in layers L_{2i-1} and L_{2i} , $1 \leq i \leq m/2$, will reverse at least $\lfloor (a^* + i)/2 \rfloor$ times.*

Proof. Consider a bad node v of L_{2i} . Node v is a source in state I . Lemma 5.4 implies that whenever v reverses in the future, it reverses all of its incident links, and therefore it remains a source. Moreover, Lemma 5.4 implies that every time that v reverses, its alpha value increases by 2. Lemma 5.5 implies that when the network stabilizes, the alpha value of v is at least $a^{\max} + i$. Since in state I the alpha value of v is a^{\min} , node v reverses at least $\lfloor (a^* + i)/2 \rfloor$ times after state I . Similarly, any other node in L_{2i} reverses at least $\lfloor (a^* + i)/2 \rfloor$ times.

Consider now a bad node w of L_{2i-1} . Node w is adjacent to at least one node u in layer L_{2i} . In state I , node u is a source, and it remains a source every time that u reverses (Lemma 5.4). Since u and w are adjacent, the reversals of u and w should alternate. This implies that node w reverses at least $\lfloor (a^* + i)/2 \rfloor$ times, since node u reverses at least $\lfloor (a^* + i)/2 \rfloor$ times. Similarly, any other node in L_{2i-1} reverses at least $\lfloor (a^* + i)/2 \rfloor$ times. \square

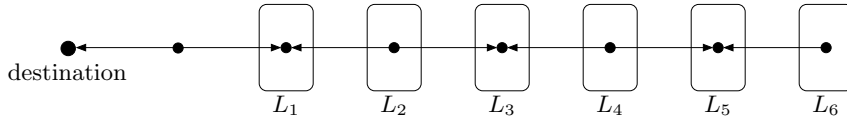


FIG. 5. Worst-case work for partial reversal: graph G_3 with $n = 6$ bad nodes.

Using Theorem 5.6 we now give worst-case graphs for work and stabilization time, which show that the upper bounds of Theorem 5.3 are tight. First, we give the lower bound on work. Consider a graph G_3 which is in state I , as described above, in which there are n bad nodes, where n is even, and there is exactly one bad node in each layer (see Figure 5). From Theorem 5.6, each node in the i th layer will reverse at least $\lfloor (a^* + \lceil i/2 \rceil)/2 \rfloor$ times before the network stabilizes. Thus, the sum of all the reversals performed by all the bad nodes is at least $\sum_{i=1}^n \lfloor (a^* + \lceil i/2 \rceil)/2 \rfloor$, which is $\Omega(n \cdot a^* + n^2)$. Thus, we have the following corollary.

COROLLARY 5.7 (work lower bound for partial reversal). *There is a graph with an initial state containing n bad nodes such that the partial reversal algorithm requires $\Omega(n \cdot a^* + n^2)$ work until stabilization.*

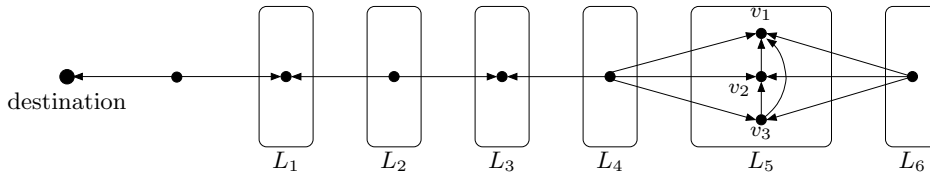


FIG. 6. Worst-case stabilization time for partial reversal: graph G_4 with $n = 8$ bad nodes, $m_1 = 6$ layers, and $m_2 = 3$ nodes in layer $m_1 - 1$.

Now we give the lower bound on time. Consider a graph G_4 in a state I as described above, in which there are n bad nodes, where $n/2$ is even. The graph consists of $m_1 = n/2 + 2$ layers. The first $m_1 - 2$ layers contain one node each, while layer $m_1 - 1$ contains $m_2 = n/2 - 1$ nodes, and layer m_1 contains 1 node. The layer $m_1 - 1$ is as follows: there are m_2 nodes v_1, v_2, \dots, v_{m_2} . Node v_i has outgoing links to all nodes v_j such that $j < i$ (see Figure 6). Note that each node in layer v_i is connected to the nodes in the adjacent layers from the specification of state I .

From Theorem 5.6, we know that each node in layer $m_1 - 1$ requires at least $k_1 = \lfloor (a^* + \lceil (m_1 - 1)/2 \rceil)/2 \rfloor$ reversals before it becomes a good node. Since layer

$m_1 - 1$ contains m_2 nodes, at least $k_1 \cdot m_2 = \Omega(n \cdot a^* + n^2)$ reversals are required before these bad nodes become good nodes. All these reversals have to be performed sequentially, since the nodes of layer $m_1 - 1$ are adjacent, and any two of these nodes cannot be sinks simultaneously. Thus, we have the following corollary.

COROLLARY 5.8 (time lower bound for partial reversal). *There is a graph with an initial state containing n bad nodes such that the partial reversal algorithm requires $\Omega(n \cdot a^* + n^2)$ time until stabilization.*

Note that Corollary 5.8 subsumes Corollary 5.7, since a lower bound on time is also a lower bound on work.

We now describe scenarios in mobile ad hoc networks which could result in the state I of graph G_4 and with arbitrary a^* value and number of nodes n . We first describe how to obtain an arbitrary a^{\max} value in a small graph. Consider a graph consisting of the destination node and two nodes w_1 and w_2 . Initially node w_1 points only to w_2 , which points to the destination; further, the alpha values of the nodes are all zero. Next, w_1 moves and gets also connected to the destination, without changing its height. Now w_2 moves and gets disconnected from the destination, but it still is connected to w_1 . However, w_2 is now a sink, and thus it performs a reversal, where its alpha value increases by one, since it has one neighbor (w_1). This scenario can be repeated an arbitrary number of times with the roles of w_1 and w_2 interchanged. This results in a state with an arbitrary value of a^{\max} .

Next, we describe how to obtain arbitrary $a^* = d$ in a state I of graph G_4 . Let $a^*(I)$ denote a^* in state I . Consider a graph H in an initial state I' in which all the nodes are good and all have alpha value equal to zero (thus, $a^*(I') = 0$). The nodes, except the destination, are divided into three components H_1 , H_2 , and H_3 . Graph H_1 consists of n nodes and is in a state isomorphic to the bad nodes in graph G_4 . Graph H_2 is a set of good nodes such that each node of H_1 is connected with an outgoing link to a good node in H_2 ; essentially, the nodes of H_1 are good because they are connected to the nodes of H_2 . Graph H_3 is a network consisting only of nodes w_1 and w_2 as described in the previous paragraph. From state I' we obtain a state I'' as follows. We let the nodes in H_3 oscillate (as described in the previous paragraph) until the alpha value of w_1 or w_2 is equal to d . Suppose that w_1 is the node that gets height d first and we stop the oscillation immediately when w_1 gets connected directly to the destination. Note that the nodes in H_1 and H_2 haven't changed their heights since I' , and therefore their alpha values remain zero in I'' (thus, $a^*(I'') = d$). Now, from state I'' we will obtain a state I as follows. The nodes in H_2 and the node w_2 disappear from the network; also, the node in the first layer of H_1 gets connected to w_1 . The resulting network configuration and state I are the same as in graph G_4 . Since the nodes in H_1 haven't changed their original alpha value (zero) and the node w_1 has the largest alpha value d , we obtain $a^*(I) = d$, as needed.⁴

6. Deterministic algorithms. In this section, we give worst-case lower bounds for the work and time needed for stabilization for any deterministic link reversal algorithm. Given an arbitrary deterministic function g , we will establish the existence of a family of graphs with initial states containing any number of $n > 2$ bad nodes which require at least $\Omega(n^2)$ work and time until stabilization. The lower bounds follow from a lower bound on the number of reversals performed by each node, which we describe next.

⁴A similar scenario could give a state I with arbitrary a^* , where a^{\min} is larger than zero. However, the state we gave with $a^{\min} = 0$ suffices for our lower bounds.

6.1. Lower bound on number of reversals. Given a graph G , we construct a state I in which we determine a lower bound on the number of reversals required by each bad node until stabilization. In particular, we decompose the bad nodes into levels and show that the node at level i reverses at least $i - 1$ times until stabilization. We then use this result to obtain lower bounds for work and time.

The construction of state I depends on a level decomposition of the network. Let s be the destination node. A node v (bad or not) is defined to be in *level* i if the (undirected graph) distance between v and s is i . Thus s is in level 0, and a node in level i is connected only with nodes in levels $i - 1$, i , and $i + 1$. Let m denote the maximum level of any node.

We now construct recursively states $I_{m+1}, I_m, I_{m-1}, \dots, I_2$ such that state $I = I_2$. The basis of the recursion is state I_{m+1} . The construction is as follows.

- In state I_{m+1} every node is good. Further, the heights of nodes in levels $1, 2, 3, \dots, m$ are in increasing order with the levels, i.e., given two vertices u, v at respective levels l_u, l_v with $l_u < l_v$, u 's height is less than v 's. An example assignment of heights in state I_{m+1} is to set a node's height to be equal to its level.
- Suppose we have constructed state I_i , where $m + 1 \geq i > 2$. We construct state I_{i-1} as follows:
 - For every node in levels $i - 1, i, \dots, m$, the height of the node in I_{i-1} is the same as its height in I_i .
 - Let \max_i denote the maximum height of a node in the destination oriented graph that is reached by an execution starting from I_i . (We note that from Corollary 3.5, \max_i does not depend on the actual execution sequence, but only on the initial state I_i .) For every node v in levels $1, 2, \dots, i - 2$, v 's height in I_{i-1} is assigned to be v 's height in I_i plus \max_i .

In the above construction, we assumed that the function g converges at a finite amount of time to a stable state starting from any initial state I_i . This assumption doesn't hurt the generality of our analysis, since if g didn't stabilize it would trivially require at least $\Omega(n^2)$ work and time, for n bad nodes, and thus our main result still holds. So, without loss of generality, we will assume that g stabilizes.

Next, we show that each state I_i , $m + 1 \geq i \geq 2$, satisfies the following properties:

- $P1_i$: The heights of nodes in levels $1, 2, 3, \dots, i - 1$ are in increasing order with the levels; i.e., given two vertices u, v at respective levels l_u, l_v with $l_u < l_v$, u 's height is less than v 's. Thus, every node in levels $1, 2, \dots, i - 1$ is a good node.
- $P2_i$: The heights of nodes in levels $i - 1, i, i + 1, \dots, m$ are in decreasing order with the levels, i.e., given two vertices u, v at respective levels l_u, l_v with $l_u < l_v$, u 's height is greater than v 's. Thus, every node in levels $i, i + 1, \dots, m$ is bad. In the case when $i = m + 1$, no node is bad.
- $P3_i$: Starting from initial state I_i , every node in level j , $j = i, i + 1, \dots, m$, reverses at least $j - i + 1$ times until stabilization. In the case when $i = m + 1$, no node reverses.

For $i = m + 1, \dots, 2$, we will now argue about the number of reversals starting from state I_i until stabilization. From Corollary 3.5, we know that all executions of a deterministic algorithm starting from the same initial state are essentially identical. In particular, the number of reversals of each node in every execution is the same.

For convenience, we consider a specific execution E_i which starts from initial

state I_i and reverses nodes in the following order: next reverse the bad node which has the smallest height in the current state. Clearly, such a node is a sink, and hence a candidate for reversal. The number of reversals of a node in any execution starting from I_i is equal to the number of reversals of the node in E_i .

LEMMA 6.1. *State I_i , $m + 1 \geq i \geq 2$, satisfies properties $P1_i$, $P2_i$, and $P3_i$.*

Proof. The proof is by induction on i . For the induction basis, state I_{m+1} clearly satisfies all the properties $P1_{m+1}$, $P2_{m+1}$, $P3_{m+1}$ from the construction of this state. Suppose now that state I_i , where $m + 1 \geq i > 2$, satisfies the respective three properties. We will show that state I_{i-1} satisfies the respective properties too. It can be easily checked that properties $P1_{i-1}$ and $P2_{i-1}$ are satisfied in I_{i-1} . We focus on property $P3_{i-1}$.

An execution is a sequence of reversals. We say that two reversals (v, h, H) and (v, h', H') of the same node in different network states are *equal* if the heights of the node and its neighbors are the same in both states, namely, $h = h'$ and $H = H'$. If two reversals are equal, then the heights of the node after the reversals are the same, since we are using a deterministic height increase function g . An execution E is said to be a prefix of an execution E' if the reversal sequence constituting E is elementwise equal to a prefix of the reversal sequence constituting E' . In Lemma 6.2, we show that E_i is a prefix of E_{i-1} .

Consider the state of the system which started in I_{i-1} , but after executing the reversals in E_i . In this state, the height of each node in levels $1, 2, \dots, i - 2$ is greater than \max_i by construction (it was greater than \max_i in I_{i-1} , and heights can never decrease). Let v be a node in level $l_v > i - 2$. After the execution segment E_i , v 's height is the same as the final height in the destination oriented graph reached from I_i , and by the definition of \max_i , this is no more than \max_i .

Thus, in the current state, the height of every node in levels $i - 1, i, \dots, m$ is less than the height of every node in levels $1, 2, \dots, i - 2$. Consider any node u in level $l_u \geq i - 1$. In the final destination oriented graph, there is a path of decreasing height from u to the destination s , and this path contains at least one node from levels $1, \dots, i - 2$. Thus, in the final state, u 's height is greater than the height of some node in levels $1, \dots, i - 2$, while it was less to begin with. This implies that u must have reversed at least once until stabilization.

In E_i , each node in level j , $j = i, i + 1, \dots, m$ has reversed at least $j - i + 1$ times. If we add an extra reversal to all nodes in levels $i - 1, i, \dots, m$, then in E_{i-1} , each node in levels $j = i - 1, i, \dots, m$ reverses at least $j - i + 2$ times, thus proving property $P3_{i-1}$. \square

LEMMA 6.2. *Execution E_i is a prefix of execution E_{i-1} .*

Proof. Executions E_i and E_{i-1} start from states I_i and I_{i-1} , respectively. Let $E_i = r_1^i, r_2^i, \dots, r_f^i$ and $E_{i-1} = r_1^{i-1}, r_2^{i-1}, \dots$. We prove by induction that $r_j^i = r_j^{i-1}$ for $j = 1, \dots, f$.

Base case. The nodes with the lowest height in I_i and I_{i-1} are the same node v , and v lies in layer m . The heights of all nodes in layer $m - 1$ are the same in I_i and I_{i-1} by construction. Thus, all of v 's neighbors have the same height in I_i and I_{i-1} , so that $r_1^i = r_1^{i-1}$.

Inductive case. Suppose that $r_1^i, r_2^i, \dots, r_l^i$ is identical to $r_1^{i-1}, r_2^{i-1}, \dots, r_l^{i-1}$ for some $l < f$. Let I_l^i and I_l^{i-1} , respectively, denote the state of the system starting from I_i after reversals $r_1^i, r_2^i, \dots, r_l^i$, and the state of the system starting from I_{i-1} after reversals $r_1^{i-1}, r_2^{i-1}, \dots, r_l^{i-1}$.

Let v be the bad node with the lowest height in I_l^i so that r_{l+1}^i reverses v . We

claim that this is also the bad node with the lowest height in I_l^{i-1} . The reason is as follows.

Node v must be at a level $l_v \geq i$, since E_i does not reverse any nodes at a lower level than i . All nodes in levels $i - 1$ or greater have the same heights in I_l^i and I_l^{i-1} , due to the induction hypothesis, and these are all less than \max_i . All nodes in levels $i - 2$ or less in I_l^{i-1} have heights greater than \max_i by construction. Thus, the bad node with the minimum height in I_l^{i-1} is also v , and its neighbors also have the same heights as in I_l^i , implying that r_{l+1}^i is the same as r_{l+1}^{i-1} . This completes the proof. \square

We are now ready to show the main result of this section.

THEOREM 6.3 (lower bound on reversals for deterministic algorithms). *Given any graph G and any height increase function g , there exists an initial state I (an assignment of heights to the nodes of G) which causes each node in level $i > 0$ to reverse at least $i - 1$ times until stabilization.*

Proof. Let m denote the maximum node level. We first construct a sequence of initial states I_{m+1}, I_m, \dots, I_2 as described above. Lemma 6.1 implies that starting from initial state I_i , each node in level $j, j \geq i$ reverses at least $j - i + 1$ times until stabilization (property $P3_i$). We take $I = I_2$. \square

6.2. Worst-case graphs. Here we give lower bounds on the work and time for any deterministic algorithm. Theorem 6.3 applies to any graph. Consider the list graph G_1 with $n + 2$ nodes, shown in Figure 3 and described in section 4.3. We construct a state I with n bad nodes as described in section 6.1. From Theorem 6.3, the lower bound for the worst-case number of reversals of any reversal algorithm on state I is the sum of the reversals of each bad node: $1 + 2 + \dots + n = \Omega(n^2)$. Thus we have the following corollary.

COROLLARY 6.4 (work lower bound for deterministic algorithms). *There is a graph with an initial state containing n bad nodes such that any deterministic reversal algorithm requires $\Omega(n^2)$ work until stabilization.*

We can derive a similar lower bound on the time needed for stabilization. We use the graph G_4 with $n + 2$ nodes, shown in Figure 4. The structure of the graph, and the parameters m_1 and m_2 , are defined as in section 5.2 with respect to $n + 1$. We construct a state I with n bad nodes as described in section 6.1. From Theorem 6.3, we know that each node in level $m_1 - 1$ of G_4 requires at least $(m_1 - 2)$ reversals before it becomes a good node. Level $m_1 - 1$ contains m_2 nodes. Therefore, at least $(m_1 - 2) \cdot m_2 = \Omega(n^2)$ reversals are required before these nodes become good nodes. All these reversals have to be performed sequentially, since the nodes of layer $m_1 - 1$ are adjacent, and no two of these nodes can be sinks simultaneously. Thus, we have the following corollary.

COROLLARY 6.5 (time lower bound for deterministic algorithms). *There is a graph with an initial state containing n bad nodes such that any deterministic reversal algorithm requires $\Omega(n^2)$ time until stabilization.*

7. Conclusions and discussion. We presented a worst-case analysis of link reversal routing algorithms in terms of work and time. We showed that for n bad nodes, the GB full reversal algorithm requires $O(n^2)$ work and time, while the partial reversal algorithm requires $O(n \cdot a^* + n^2)$ work and time. The above bounds are tight in the worst case. Our analysis for the full reversal is exact. For any network, we present a decomposition of the bad nodes in the initial state into *layers*, which allows us to predict *exactly* the work performed by each node in *any* distributed execution.

Furthermore, we show that for any deterministic reversal algorithm on a given

graph, there exists an assignment of heights to all the bad nodes in the graph such that if a bad node d hops away from its closest good node, then it has to reverse d times before stabilization. Using this, we show that there exist networks and initial states with n bad nodes such that the algorithm needs $\Omega(n^2)$ work and time until stabilization. As a consequence, from the worst-case perspective, the full reversal algorithm is work and time optimal, while the partial reversal algorithm is not. Since a^* can grow arbitrarily large, the full reversal algorithm outperforms the partial reversal algorithm in the worst case.

Since it is known that partial reversal performs better than full reversal in some cases, it would be interesting to find a variation of the partial reversal algorithm, which is as good as full reversal in the worst case. Another research problem is to analyze the average performance of link reversal algorithms. It would be also interesting to extend our analysis to nondeterministic algorithms, such as randomized algorithms, in which the new height of a sink is some randomized function of the neighbors' heights.

Acknowledgments. We thank the reviewers for their valuable comments and suggestions. We also thank Srikanth Surapaneni for helping in the preparation of an earlier version of this paper.

REFERENCES

- [1] J. BROCH, D. A. MALTZ, D. B. JOHNSON, Y.-C. HU, AND J. JETCHEVA, *A performance comparison of multi-hop wireless ad hoc network routing protocols*, in Proceedings of the Fourth Annual ACM/IEEE International Conference on Mobile Computing and Networking (MOBICOM), 1998, ACM, New York, pp. 85–97.
- [2] C. BUSCH, S. SURAPANENI, AND S. TIRTHAPURA, *Analysis of link reversal routing algorithms for mobile ad hoc networks*, in Proceedings of the Fifteenth Annual ACM Symposium on Parallelism in Algorithms and Architectures (SPAA), 2003, pp. 210–219.
- [3] I. CHATZIGIANNAKIS, E. KAL TSA, AND S. NIKOLETSEAS, *On the effect of user mobility and density on the performance of routing protocols for ad-hoc mobile networks*, *Wireless Communication and Mobile Computing*, 4 (2004), pp. 609–621.
- [4] I. CHATZIGIANNAKIS, S. NIKOLETSEAS, AND P. SPIRAKIS, *Distributed communication algorithms for ad-hoc mobile networks*, *J. Parallel Distrib. Comput.*, 63 (2003), pp. 58–74.
- [5] M. S. CORSON AND A. EPHREIMIDES, *A distributed routing algorithm for mobile radio networks*, in Proceedings of the IEEE Military Communications Conference (MILCOM), 1989, pp. 210–213.
- [6] M. S. CORSON AND A. EPHREIMIDES, *A distributed routing algorithm for mobile wireless networks*, *ACM/Baltzer Wireless Networks J.*, 1 (1995), pp. 61–82.
- [7] E. M. GAFNI AND D. P. BERTSEKAS, *Distributed algorithms for generating loop-free routes in networks with frequently changing topology*, *IEEE Trans. Comm.*, 29 (1981), pp. 11–18.
- [8] C. INTANAGONWIWAT, R. GOVINDAN, D. ESTRIN, J. HEIDEMANN, AND F. SILVA, *Directed diffusion for wireless sensor networking*, *IEEE/ACM Trans. Networking (TON)*, 11 (2003), pp. 2–16.
- [9] N. MALPANI, J. L. WELCH, AND N. VAIDYA, *Leader election algorithms for mobile ad hoc networks*, in Proceedings of the Fourth Annual ACM International Workshop on Discrete Algorithms and Methods for Mobile Computing and Communication (DIAL-M), 2000, pp. 96–103.
- [10] V. D. PARK AND M. S. CORSON, *A highly adaptive distributed routing algorithm for mobile wireless networks*, in Proceedings of the Sixteenth Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM), 1997, pp. 1405–1413.
- [11] V. S. PARK AND M. S. CORSON, *A performance comparison of the temporally-ordered routing algorithm and ideal link-state routing*, in Proceedings of the Third Annual IEEE International Symposium on Computers and Communications, 1998, pp. 592–598.
- [12] C. E. PERKINS, *Ad Hoc Networking*, Addison–Wesley, Reading, MA, 2000.
- [13] R. RAJARAMAN, *Topology control and routing in ad hoc networks: A survey*, *SIGACT News*, 33 (2002), pp. 60–73.
- [14] S. R. DAS, R. CASTANEDA, Y. JIANGTAO, AND R. SENGUPTA, *Comparative performance evaluation of routing protocols for mobile, ad hoc networks*, in Proceedings of the Seventh Annual IEEE International Conference on Computer Communications and Networks (IC3N), 1998, pp. 153–161.

TWO-WAY CHAINING WITH REASSIGNMENT*

KETAN DALAL[†], LUC DEVROYE[†], EBRAHIM MALALLA[†], AND ERIN MCLEISH[†]

Abstract. We present an algorithm for hashing $\lfloor \alpha n \rfloor$ elements into a table with n separate chains that requires $O(1)$ deterministic worst-case insert time and $O(1)$ expected worst-case search time for constant α . We exploit the connection between two-way chaining and random graph theory in our techniques.

Key words. hashing, two-way chaining, worst-case search time, random graphs, probabilistic analysis of algorithms

AMS subject classifications. 68Q25, 68M20, 68P10, 60G99, 05C80

DOI. 10.1137/S0097539704443240

1. Introduction. In classical uniform hashing with chaining, a set of s keys is inserted into a hash table with n separate chains (or linked lists) via a uniform hash function. The insertion time is constant, and the average search time is proportional to the load factor of the hash table $\alpha := s/n$. However, even for a constant load factor, the worst-case search time (the length of the longest chain) is asymptotic to $\log n / \log \log n$ in probability [18, 27].

Azar et al. [3] suggested a novel approach called the *greedy two-way chaining* paradigm. It uses two independent uniform hash functions to insert the keys, where each key is inserted *on-line* into the shorter chain, with ties broken randomly. The insertion time is still constant, while the average search time cannot be more than twice the average search time of classical uniform hashing. However, the expected maximum search time is only $2 \log_2 \log n + 2\alpha + O(1)$ [3, 4, 24]. The two-way chaining paradigm has been effectively used to derive many efficient algorithms [5, 6, 7]. A further variant of on-line two-way chaining [28] improves the maximum search time by a constant factor.

On the other hand, one can show that the *off-line* version of two-way chaining, where all the hashing values of the keys are known in advance, yields better worst-case performance [3, 8, 25]. Czumaj and Stemann [8] proved that if $s \leq 1.67545943 \dots \times n$, one can find an assignment for the keys such that the maximum chain length is at most 2 w.h.p. (with high probability, i.e., with probability tending to one as $n \rightarrow \infty$). In general, for any integer $k \geq 2$, it is known [23] that there is a threshold $c_k \sim k$ such that if $s \leq c_k n$, one can assign the keys such that the maximum chain length is at most k w.h.p. The insertion time, however, is proportional to s . This shows that there is a large gap between the worst-case performances of the on-line and off-line versions of two-way chaining. One wonders if it is possible to design an efficient on-line two-way chaining algorithm whose worst-case search time is close enough to its off-line one, while preserving constant insertion time and $O(\alpha)$ average search time. Our goal here is to obtain constant expected maximum search and deterministic $O(1)$

*Received by the editors April 22, 2004; accepted for publication (in revised form) May 9, 2005; published electronically October 7, 2005. This research was supported by NSERC grant A3456, by NSERC Centre of Excellence IRIS grant “Learning machines,” and by the National Science Foundation Graduate Research Fellowship.

<http://www.siam.org/journals/sicomp/35-2/44324.html>

[†]School of Computer Science, McGill University, Montreal, QC, Canada (kdalal@cs.mcgill.ca, luc@cs.mcgill.ca, emal-a@cs.mcgill.ca, mcleish@cs.mcgill.ca).

insertion times when the load factor of the hash table is constant.

Many hashing schemes that achieve constant worst-case search time have been developed [11, 12, 13, 16]. However, these schemes use a large number of hash functions, sometimes employ rehashing techniques, and have insertion times that are constant only in an expected amortized sense. Closest to our work is a new hashing scheme called cuckoo hashing [26, 10], which utilizes the two-choice paradigm to improve the worst-case performance but also relies on the idea of reallocation of the inserted keys. It inserts n keys into a hash table that is partitioned into two parts, each of size $\lceil (1 + \epsilon)n \rceil$, for some constant $\epsilon > 0$. It uses two independent hash functions chosen from an $O(\log n)$ -universal class—one function only for each subtable. Each key is hashed initially by the first function to a cell in the first subtable. If the cell is full, then the new key is inserted there anyway, and the old key is kicked out to the second subtable to be hashed by the second function. The same rule is applied in the second subtable. Keys are moved back and forth until a key moves to an empty location or a limit of $O(\log n)$ moves is reached. When the limit is reached, new independent hash functions are chosen, and the whole table is rehashed. The worst-case search time is at most two, but the insertion time is constant only in an amortized expected sense. An off-line and static version of this algorithm previously appeared in [25].

In this paper, we present a two-way chaining algorithm that is close to cuckoo hashing but achieves constant worst-case insertion time, deterministically, and constant worst-case search time asymptotically almost surely, when the load factor is constant. The space consumption is also linear. The idea is based on the structure of a random multigraph, a key reassignment technique, and a deamortization method. The algorithm is divided into stages where, at each stage, the hash table is modeled by a random graph with n vertices representing the chains and m edges denoting the keys inserted during the stage. Inserting keys into chains corresponds to orienting edges toward vertices. Our goal then is to minimize the maximum out-degree. This model has been used earlier to analyze the off-line version of two-way chaining [8]. When the graph is a forest, it is easy to orient the edges such that the maximum out-degree is one. In order to keep the maximum out-degree as low as possible, some edges need to be reoriented when two trees are joined during the hashing process, and this means that the corresponding keys also need to be reassigned. Furthermore, cycles could occur in the random graph. Since the hashing process is on-line, we use a queue to control the orientation process, thereby ensuring that every insertion operation takes only a constant time of work. This leads us to the elegant deamortization method introduced by Gajewska and Tarjan [17]. In the next section we describe the algorithm precisely and ensure that an insert takes $O(1)$ deterministic worst-case time. We analyze the worst-case search time in section 3.

2. The algorithm. We start by presenting a simplified algorithm that requires $\omega(1)$ insertion time in the worst case and then use a standard deamortization trick to reduce the insertion time to $O(1)$.

Our algorithm inserts $s = \lfloor \alpha n \rfloor$ keys into a hash table \mathcal{T} with n separate chains (implemented as doubly linked lists) denoted by $\mathcal{T}[1], \dots, \mathcal{T}[n]$ by using two independent uniform hash functions f and g . We assume throughout that f and g map the space of the keys to $\{1, \dots, n\}$ such that if x_1, \dots, x_s are different keys, then $f(x_1), g(x_1), \dots, f(x_s), g(x_s)$ are independent and uniformly distributed on $\{1, \dots, n\}$. So, a key x is inserted into one of the chains $\mathcal{T}[f(x)]$ or $\mathcal{T}[g(x)]$. To search for any key, we examine only the two possible hashing chains available to it. Thus, the worst-case search time is at most twice the length of the longest chain plus the time needed to

compute the hashing values. For simplicity, we ignore the time for evaluating the hash functions.

The hashing process is described as follows (see Figure 2.1). In addition to the hash table, the algorithm maintains a directed graph $\mathcal{G}(V, A)$, where V is a set of n vertices and A is a set of arcs. Each vertex of \mathcal{G} corresponds to a chain of \mathcal{T} . An arc $\langle u, v \rangle \in A$ implies that there exists some key x whose hash values are u and v . The direction of this arc indicates that x is located in the chain corresponding to u , i.e., $\mathcal{T}[u]$. (With some abuse of notation, we will refer to an edge (u, v) of \mathcal{G} to indicate either $\langle u, v \rangle$ or $\langle v, u \rangle$.) In addition, let \mathbf{X} be a pointer to the linked-list node that contains x , and let $*\mathbf{X}$ be the node itself. An important property of the arcs is that they correspond to a *subset* of the keys contained in \mathcal{T} ; i.e., some keys are *dropped*.

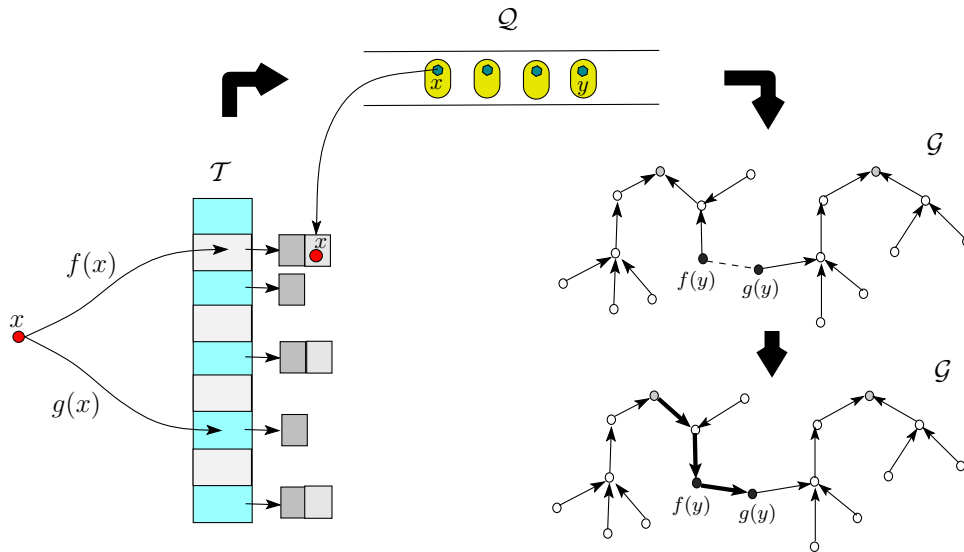


FIG. 2.1. Upon arrival of key x , it is inserted into $\mathcal{T}[f(x)]$, and then a request for adding a corresponding arc is appended to the queue \mathcal{Q} . Next, κ extra units of work are performed to process the requests at the front of the queue. The figure also illustrates the **ReverseRoot** operation and the insertion of $\langle f(y), g(y) \rangle$.

To insert a key x into \mathcal{T} , see Pseudocode 1. Notice that initially any key x is always inserted into the chain $\mathcal{T}[f(x)]$. However, during the hashing process the key x may be reassigned back and forth between the two chains $\mathcal{T}[f(x)]$ and $\mathcal{T}[g(x)]$. This could happen by edge reversals as is shown, e.g., in Pseudocode 5.

Pseudocode 1 Insert(x)

- 1: $u \leftarrow f(x)$
 - 2: $v \leftarrow g(x)$
 - 3: Create new linked-list node, $*\mathbf{X}$ containing key x
 - 4: Insert $*\mathbf{X}$ into $\mathcal{T}[u]$.
 - 5: **UpdateGraph**(u, v, \mathbf{X}).
-

The **UpdateGraph** operation enforces that \mathcal{G} is acyclic and that every vertex has out-degree at most one. This means that the graph is simply a forest of parent-pointer trees. We represent the graph in an array, where the array element corresponding to

a vertex u contains a parent-pointer $P[u]$ and a pointer $X[u]$ to the linked-list node. Thus the array element for vertex u represents the arc, $\langle u, P[u] \rangle$. The `UpdateGraph` operation is described in Pseudocode 2. Note that in some cases, no edge is inserted into \mathcal{G} at all.

Pseudocode 2 `UpdateGraph(u, v, X)`

```

1:  $r_1 \leftarrow \text{FindRoot}(u)$ 
2:  $r_2 \leftarrow \text{FindRoot}(v)$ 
3: if  $r_1 \neq r_2$  then
4:   ReverseRoot(u)
5:   Link(u, v, X).
6: end if

```

The `FindRoot(u)` operation starts at u and takes parent pointers until the root is found. See Pseudocode 3.

Pseudocode 3 `FindRoot(u)`

```

1:  $r_1 \leftarrow u$ 
2: while  $P[r_1] \neq \text{nil}$  do
3:    $r_1 \leftarrow P[r_1]$ 
4: end while
5: return  $r_1$ 

```

The `Link(u, v, X)` operation creates the arc $\langle u, v \rangle$ and updates \mathcal{T} accordingly. Recall that the chains of \mathcal{T} are implemented as doubly linked lists. Thus, the updates to \mathcal{T} can be performed in $O(1)$ time. Additionally, `Link(u, v, X)` requires that u be a root. See Pseudocode 4.

Pseudocode 4 `Link(u, v, X)`

```

1:  $P[u] \leftarrow v$ 
2:  $X[u] \leftarrow X$ 
3: Move  $*X$  to  $\mathcal{T}[u]$ 

```

The `ReverseRoot(u)` function reverses the sequence of pointers from u to the root of u 's component. See Pseudocode 5. At the end of this operation, u is the new root of u 's component. Note that each reversal will update \mathcal{T} as part of the `Link` operation.

The following two facts are easy to see.

LEMMA 1. *At any point in time, the graph \mathcal{G} is a forest of parent-pointer trees with no self-loops or multiple edges.*

LEMMA 2. *The `Insert` operation requires worst-case time not exceeding $4M + O(1)$, where M is the maximum size of any tree in \mathcal{G} .*

We use two simple techniques to reduce the cost of the insertions. First, to reduce the size of the trees, after each $m = \lfloor \beta n \rfloor$ inserts, where $\beta < 1/2$ is some constant to be picked later, we destroy \mathcal{G} , which amounts to simply zeroing the array representation of \mathcal{G} .

Next, following [17], we use a queue \mathcal{Q} (implemented as a linked list) to defer some work to reduce the cost per insert. We define κ to be a (constant) parameter of the algorithm that indicates the maximum number of operations that may take place as part of each insert to process work items in \mathcal{Q} . (The dependence of κ on β will

Pseudocode 5 ReverseRoot(u)**Ensure:** $P[u] = \text{nil}$

```

1: if  $P[u] \neq \text{nil}$  then
2:   ReverseRoot( $P[u]$ )
3:   Temp  $\leftarrow P[u]$ 
4:    $P[u] \leftarrow \text{nil}$ 
5:   Link(Temp,  $u$ ,  $X[u]$ )
6: end if

```

be made clear later on.) The algorithm is modified as follows. After every hash table insert, where a new list node is created for key x and inserted into the chain $\mathcal{T}[f(x)]$, we append a *graph-insert request* to the *end* of the queue, \mathcal{Q} . This is a request for adding the edge $(f(x), g(x))$ to the graph \mathcal{G} and orienting that edge appropriately. Additionally, κ extra units of work will be performed on the request at the front of the queue, where one unit of work can be used to traverse or reverse an edge in the graph. Once the request is completed, it is deleted from the queue, and the remaining time is spent on the next element of the queue until either \mathcal{Q} is empty or the κ available time steps are depleted.

To combine both techniques, we keep an extra graph structure \mathcal{G}' that is zeroed incrementally as elements are inserted into \mathcal{G} . After $\lfloor \beta n \rfloor$ inserts, we will simply swap \mathcal{G} and \mathcal{G}' . This allows us to reduce the cost of zeroing the graph structure every $\lfloor \beta n \rfloor$ inserts. There are some minor technicalities in implementing this approach and the complete pseudocode for **ConstantInsert** is given in Pseudocode 6 in the appendix. Essentially, the approach is to break down the operation of **UpdateGraph** into constant time pieces.

We will write $\text{HASH}(n, s, m, \kappa)$ to refer to this modified process of hashing s keys into a hash table of size n , where m keys are inserted at each stage and κ is the constant parameter mentioned above. We omit the details for initializing the data structures and keeping track of an edge counter.

LEMMA 3. *Using the queue, the **ConstantInsert** operation requires time proportional to κ .*

The deferral described above creates a potential inconsistency between the state of the hash table and the state of the graph. Specifically, at the completion of a particular insertion request, the graph represents the state of the hash table at an earlier point in time, i.e., prior to the requests that are still pending in the queue. Only when the queue is empty will the graph represent the state of the hash table. Additionally, because some requests are dropped and, on occasion, the graph is destroyed, the graph may contain only a subset of the state of the hash table.

3. The worst-case search time. We shall prove the following theorem.

THEOREM 1. *There is a constant $\kappa > 0$ such that at any point in time during the hashing process $\text{HASH}(n, s, m, \kappa)$, where $n, s, m \in \mathbb{N}$, and $m = \lfloor \beta n \rfloor \leq s = O(n \log n)$, for some constant $\beta < 1/2$, the maximum search time is at most $2 \lceil s/m \rceil + 6$ w.h.p.*

The theorem confirms that if the load factor of the hash table $s/n = O(1)$, then asymptotically almost surely the maximum search time is constant. Since there is a trivial lower bound of $2s/n$, we see that we are roughly within $1/\beta$ of the best possible, recalling that β can be picked arbitrarily close to $1/2$. Before we proceed with the proof, we need some facts. We write $\text{Bin}(n, p)$ to denote a binomial random variable with parameters $n \in \mathbb{N}$ and $p \in [0, 1]$. We recall the following binomial tail inequalities.

LEMMA 4 (Angluin and Valiant [2]). For $n \in \mathbb{N}$, $p \in [0, 1]$, and constant $\epsilon \in (0, 1)$, we have

$$\mathbb{P} \{ \text{Bin}(n, p) \geq (1 + \epsilon)np \} \leq e^{-np\epsilon^2/3}$$

and

$$\mathbb{P} \{ \text{Bin}(n, p) \leq (1 - \epsilon)np \} \leq e^{-np\epsilon^2/2}.$$

Let $\mathbb{G}(n, m)$ denote a random graph with n vertices and m multiedges that may include loops, where each edge connects two vertices chosen—one after another—independently and uniformly at random, with replacement, from the set of all n vertices. This means that any loop is realized with probability $1/n^2$ and any undirected nonloop edge is realized with probability $2/n^2$. Recall that the classical model $G(n, p)$ of Erdős [14] and Erdős and Rényi [15] has no loops or multiedges, and each edge is realized with a fixed probability $p \in (0, 1)$. Throughout, we write $[n]$ to denote the set $\{1, \dots, n\}$.

LEMMA 5. Let $\mathcal{C}(u)$ be the number of vertices in the connected component containing a fixed vertex u from the random graph $\mathbb{G}(n, m)$, where $n \in \mathbb{N}$, and $m = \lfloor \beta n \rfloor$, for some constant $\beta < 1/2$. Then for any $t \in [n]$, we have $\mathbb{P} \{ \mathcal{C}(u) > t \} \leq 2e^{-\gamma t}$, where $\gamma = (1/2 - \beta)^2 / (2 + 4\beta)$. Thus, if M is the size of the largest connected component, then $\mathbb{P} \{ M > (1 + \epsilon)\gamma^{-1} \log n \} \leq 2n^{-\epsilon}$ for any fixed $\epsilon > 0$.

Proof. We first need to distinguish between the components of the classical $G(n, p)$ and those of $\mathbb{G}(n, m)$. Let $\mathcal{R}_p(u)$ denote the number of vertices in the component containing vertex u in $G(n, p)$ and use $C_k(u)$ for our model $\mathbb{G}(n, k)$. Next let $|G(n, p)|$ be the number of edges in $G(n, p)$. Notice that

$$\mathbb{P} \{ \mathcal{R}_p(u) > t \mid |G(n, p)| = k \} \geq \mathbb{P} \{ C_k(u) > t \}$$

because, conditional on having k edges, components in $G(n, p)$ are stochastically larger than those in $\mathbb{G}(n, k)$ since the latter includes multiedges and loops. This leads to the following relationship between $\mathcal{R}_p(u)$ and $C_m(t)$:

$$\begin{aligned} \mathbb{P} \{ \mathcal{R}_p(u) > t \} &= \sum_k \mathbb{P} \{ \mathcal{R}_p(u) > t \mid |G(n, p)| = k \} \mathbb{P} \{ |G(n, p)| = k \} \\ &\geq \sum_k \mathbb{P} \{ C_k(u) > t \} \mathbb{P} \{ |G(n, p)| = k \} \\ &\geq \sum_{k \geq m} \mathbb{P} \{ C_k(u) > t \} \mathbb{P} \{ |G(n, p)| = k \} \\ &\geq \sum_{k \geq m} \mathbb{P} \{ C_m(u) > t \} \mathbb{P} \{ |G(n, p)| = k \} \\ &= \mathbb{P} \{ C_m(u) > t \} \mathbb{P} \{ |G(n, p)| \geq m \} \\ &\geq \mathbb{P} \{ C_m(u) > t \} - \mathbb{P} \{ |G(n, p)| < m \}. \end{aligned}$$

Thus,

$$\mathbb{P} \{ C_m(u) > t \} \leq \mathbb{P} \{ \mathcal{R}_p(u) > t \} + \mathbb{P} \{ |G(n, p)| < m \}.$$

Bounding $\mathbb{P} \{ \mathcal{R}_p(u) > t \}$ in the classical model is done in the usual manner; see, for example, Janson [21]. Imagine that u 's component grows out from vertex u , picking

up neighbors according to a binomial distribution. This certainly overestimates the number of vertices in u 's component. Now suppose that the component containing vertex u contains more than t vertices. This implies that the sum of t binomial random variables is at least t . Denote these random variables by X_i for $i = 1, \dots, t$, and for an upper bound on $|\mathcal{R}_p(u)|$ assume that each X_i is distributed as $\text{Bin}(n, p)$ and that they are independent. The sum of t independent $\text{Bin}(n, p)$ is itself a $\text{Bin}(nt, p)$ random variable. So we have that

$$\mathbb{P}\{|\mathcal{R}_p(u)| > t\} \leq \mathbb{P}\{\text{Bin}(nt, p) \geq t\}.$$

Using Lemma 4, with $p = \frac{\beta+1/2}{n}$ and $\epsilon = \frac{1/2-\beta}{1/2+\beta} \in (0, 1)$, we get

$$\mathbb{P}\left\{\text{Bin}\left(nt, \frac{\beta+1/2}{n}\right) \geq t\right\} \leq \exp\left(\frac{-t(1/2-\beta)^2}{3(1/2+\beta)}\right).$$

Now it only remains to bound $\mathbb{P}\{|G(n, p)| < m\}$. Notice that $|G(n, p)|$ is distributed as $\text{Bin}(N, p)$, where $N = \binom{n}{2}$, and

$$\begin{aligned} m-1 &\leq \beta n - 1 = (n-1)(\beta/2 + 1/4) - (n/4 - \beta n/2 + 1 - \beta/2 - 1/4) \\ &\leq Np - x, \end{aligned}$$

where $x = (1/4 - \beta/2)n$. Using the lower tail bound of Lemma 4, we get

$$\begin{aligned} \mathbb{P}\{|G(n, p)| < m\} &= \mathbb{P}\{\text{Bin}(N, p) \leq m-1\} \\ &\leq \mathbb{P}\{\text{Bin}(N, p) \leq Np - x\} \\ &\leq \exp\left(\frac{-x^2}{2Np}\right) = \exp\left(\frac{-n^2(1/2-\beta)^2}{4(n-1)(1/2+\beta)}\right) \leq \exp\left(\frac{-n(1/2-\beta)^2}{4(1/2+\beta)}\right). \end{aligned}$$

Putting everything together, we see that the resulting bound for the component size in our model is

$$\mathbb{P}\{C(u) > t\} \leq \exp\left(\frac{-n(1/2-\beta)^2}{4(1/2+\beta)}\right) + \exp\left(\frac{-t(1/2-\beta)^2}{3(1/2+\beta)}\right).$$

Since the component size $t \leq n$,

$$\mathbb{P}\{C(u) > t\} \leq 2 \exp\left(\frac{-t(1/2-\beta)^2}{4(1/2+\beta)}\right) = 2e^{-\gamma t},$$

where $\gamma = (1/2 - \beta)^2 / (2 + 4\beta)$. □

The next lemma, which is included to make the paper self-contained, ensures that the asymptotic structure of $\mathbb{G}(n, m)$ is not complex when $m < n/2$. Further details can be found in [20]. An edge is said to *complete* a cycle if both of its vertices are chosen from the same connected component before its insertion.

LEMMA 6. *Let $n \in \mathbb{N}$, and $m = \lfloor \beta n \rfloor$, for some constant $\beta < 1/2$. In the random graph $\mathbb{G}(n, m)$, the expected number of edges that complete cycles is $O(\log n)$. Furthermore, the probability that $\mathbb{G}(n, m)$ contains a connected component with more than one cycle is $o(1 / \log n)$.*

Proof. Let Y_i be the number of edges that complete cycles in $\mathbb{G}(n, m)$ after $(i-1)$ edges have been inserted. Let D_i be the event that the i th edge completes a cycle. Let

M_i be the random variable corresponding to the size of the largest component after $(i - 1)$ edges have been inserted. Using the fact that the sequence $\{M_i\}$ is increasing,

$$\begin{aligned} \mathbf{E}[Y_{m+1}] &= \sum_{i=1}^m \mathbb{P}\{D_i\} \\ &= \mathbf{E}\left[\sum_{i=1}^m \mathbb{I}_{[D_i]}\right] = \mathbf{E}\left[\mathbf{E}\left[\sum_{i=1}^m \mathbb{I}_{[D_i]} \mid M_i\right]\right] \\ &\leq \sum_{i=1}^m \mathbf{E}[M_i/n] \leq \mathbf{E}[M_{m+1}] \\ &\leq \frac{2}{\gamma} \log n + m \mathbb{P}\left\{M_{m+1} > \frac{2}{\gamma} \log n\right\} \\ &= O(\log n), \end{aligned}$$

which follows from Lemma 5. Next, we show that it is unlikely for a component to contain more than one cycle.

Let A_i be the event that $M_i \leq a \log n$, where a is chosen such that $\mathbb{P}\{A_{m+1}^c\} = O(1/n)$. Let B_i be the event that $Y_i \leq \log^3 n$. Using $\mathbf{E}[Y_{m+1}] = O(\log n)$ and Markov’s inequality, we have $\mathbb{P}\{B_{m+1}^c\} = O(1/\log^2 n)$. Let C_i be the event that the i th edge causes the creation of a component that contains two cycles. Equivalently, C_i is the event that the i th edge connects two (not necessarily distinct) cyclic components. Treating these events as sets, we obtain

$$\begin{aligned} C_i &= (C_i \cap A_i \cap B_i) \cup (C_i \cap B_i^c \cap A_i) \cup (C_i \cap A_i^c) \\ &\subseteq (C_i \cap A_i \cap B_i) \cup B_i^c \cup A_i^c. \end{aligned}$$

Since A_i^c and B_i^c are increasing events, then $\bigcup_{i=1}^{m+1} A_i^c = A_{m+1}^c$, and similarly, $\bigcup_{i=1}^{m+1} B_i^c = B_{m+1}^c$. Consequently,

$$\begin{aligned} \mathbb{P}\left\{\bigcup_i C_i\right\} &\leq \left(\sum_{i=1}^m \mathbb{P}\{C_i, A_i, B_i\}\right) + \mathbb{P}\{B_{m+1}^c\} + \mathbb{P}\{A_{m+1}^c\} \\ &\leq \left(\sum_{i=1}^m \mathbb{P}\{C_i \mid A_i, B_i\}\right) + O\left(\frac{1}{\log^2 n}\right) + O\left(\frac{1}{n}\right) \\ &\leq m \left(\frac{(a \log n)(\log^3 n)}{n}\right)^2 + O\left(\frac{1}{\log^2 n}\right) = O\left(\frac{1}{\log^2 n}\right), \end{aligned}$$

as the maximum number of “bad” vertices that the i th edge can choose from is at most Y_i times M_i , which is not more than $a \log^4 n$. \square

Recall that the hashing process $\text{HASH}(n, s, m, \kappa)$ is divided into $N := \lceil s/m \rceil$ different stages, where at each stage $m \lfloor \beta n \rfloor$ keys are inserted into the hash table. Consider only the first stage. Recall that the graph \mathcal{G} does not fully represent the hash table because, first, the keys are inserted into the hash table without any delay, while the edges are queued in \mathcal{Q} for what might be a long time before they are actually inserted into the graph \mathcal{G} , and, second, any edge that completes a cycle is not added to the graph. For convenience, we will write $\mathcal{G}(m)$ to denote the graph \mathcal{G} at the end of the first stage, i.e., after having fully processed m edges, and write $\mathcal{G}(m)^+$ to denote the complete graph of $\mathcal{G}(m)$ plus all dropped edges that complete cycles. Observe

that the undirected version of the graph $\mathcal{G}(m)^+$ is stochastically equivalent to the random graph $\mathbb{G}(n, m)$. The following lemma shows that the dropped edges of the whole hashing process are disjoint. For any multiset of edges \mathcal{E} , and for any vertex u in the graph, let $\mathcal{V}(u, \mathcal{E})$ be the multiset of all vertices v such that $(u, v) \in \mathcal{E}$. Let $\deg(u, \mathcal{E}) = |\mathcal{V}(u, \mathcal{E})|$ be the degree of u in \mathcal{E} .

LEMMA 7. *Let \mathcal{D} be the multiset of dropped edges during all stages of the hashing process $\text{HASH}(n, s, m, \kappa)$, where n, s , and m are as defined in Theorem 1. Then $\max_u \deg(u, \mathcal{D}) = 1$ w.h.p.*

Proof. Recall that the number of stages is $N := \lceil s/m \rceil = O(\log n)$. For $i = 1, \dots, N$, let \mathcal{D}_i be the multiset of all dropped edges in stage i . Since the dropped edges are the ones that complete cycles, then Lemma 6 implies that $\mathbf{E}[|\mathcal{D}_1|] = O(\log n)$, and

$$\mathbb{P} \left\{ \max_u \deg(u, \mathcal{D}_1) > 1 \right\} = o(1 / \log n),$$

because $\deg(u, \mathcal{D}_1) > 1$ implies that the component containing u has more than one cycle. Clearly, $\sum_u \deg(u, \mathcal{D}_1) \leq 2|\mathcal{D}_1|$. Since we have n vertices in the graph, then

$$\mathbf{E}[\deg(u, \mathcal{D}_1)] = \mathbf{E}[\mathbf{E}[\deg(u, \mathcal{D}_1) \mid |\mathcal{D}_1|]] \leq 2\mathbf{E}[|\mathcal{D}_1|] / n.$$

For $i \neq j$, let $A_{i,j}$ be the event that there is a vertex u appearing in two dropped edges in \mathcal{D}_i and \mathcal{D}_j ; i.e., there are vertices v and w such that $(u, v) \in \mathcal{D}_i$ and $(u, w) \in \mathcal{D}_j$. Since $\mathcal{D}_1, \dots, \mathcal{D}_N$ are independent and identically distributed, we have

$$\begin{aligned} \mathbb{P} \left\{ \max_u \deg(u, \mathcal{D}) > 1 \right\} &\leq N \mathbb{P} \left\{ \max_u \deg(u, \mathcal{D}_1) > 1 \right\} + \binom{N}{2} \mathbb{P} \{A_{1,2}\} \\ &\leq o(1) + N^2 n (\mathbb{P} \{ \deg(u, \mathcal{D}_1) \geq 1 \})^2 \\ &\leq o(1) + N^2 n (\mathbf{E}[\deg(u, \mathcal{D}_1)])^2 \\ &\leq o(1) + N^2 n \left(\frac{2\mathbf{E}[|\mathcal{D}_1|]}{n} \right)^2 \\ &= o(1) + O \left(\frac{(\log n)^4}{n} \right) = o(1). \quad \square \end{aligned}$$

Finally, we recall the following inequality.

LEMMA 8 (Hoeffding [19]). *Let S be a set of m balls, where ball i has a value x_i . Let X_1, \dots, X_ν be the values of ν balls chosen from S independently and uniformly at random without replacement. Let Y_1, \dots, Y_ν be the values of ν balls chosen from S independently and uniformly at random with replacement. Then for any continuous convex function f , we have*

$$\mathbf{E} \left[f \left(\sum_{i=1}^{\nu} X_i \right) \right] \leq \mathbf{E} \left[f \left(\sum_{i=1}^{\nu} Y_i \right) \right].$$

Proof of Theorem 1. Recall that $\mathcal{G}(m)$ denotes the graph \mathcal{G} at the end of the first stage, and $\mathcal{G}(m)^+$ denotes the complete graph of $\mathcal{G}(m)$ plus all dropped edges that complete cycles. First of all, Lemma 7 says that the vertices of all dropped edges of the whole hashing process $\text{HASH}(n, s, m, \kappa)$ are distinct w.h.p. That is, the corresponding keys of these edges are inserted into distinct chains, or in other words, any chain harbors at most one key that corresponds to a dropped edge. Therefore, upon termination of the hashing process, the dropped edges may contribute at most one to the maximum chain length.

We shall prove that w.h.p. during any interval of time (measured with respect to the number of keys inserted) of length $\nu := \lfloor n^{1/4} \rfloor$ (at any stage), the queue \mathcal{Q} must be empty at least once, and no connected component is chosen more than twice. This means that w.h.p. any set of requests that exist in the queue at some point in time could have at most two requests for inserting edges in the same connected component; that is, we could insert at most two keys into the same chain before the final positions of the related keys are corrected. Assume this is true for the time being. Then clearly the length of any chain in the hash table during the first stage is not more than the out-degree of the corresponding vertex in \mathcal{G} plus three: one for any possible dropped edge contribution, and two for the two requests in the queue that have chosen the same component. However, the maximum out-degree of \mathcal{G} is ensured to be one all the time. Hence, the maximum chain length at any point in time during the first stage is at most four w.h.p. Since we follow the same strategy at each stage, it is not difficult to see that the chain length increases by at most one per stage. Note that over all stages, each chain has at most one dropped edge contribution, and a single component is present at most twice in the queue. Consequently, the maximum chain length at any point in time during the hashing process is at most $\lceil s/m \rceil + 3$ w.h.p., and hence the worst-case search time is at most $2 \lceil s/m \rceil + 6$ w.h.p., as we have to search two chains for each key.

Now we prove our assumption. We say “at time i ” to mean at the insertion time of the i th key. Thus, an interval of time of length ν means a period of time into which exactly ν keys are inserted. Let M be the size of the largest connected component in $\mathcal{G}(m)$. Since $\beta < 1/2$, then by Lemma 5, $\mathbb{P}\{M > 2\gamma^{-1} \log n\} = O(1/n)$.

Let A_{ij} be the event that the j th component is hit by the i th request and is hit at least twice more in the subsequent $\nu - 1$ requests. Note that when the insertion request at the front of the queue is being processed, the j th component is fixed. Let $A = \bigcup_{i,j} A_{ij}$. Let M_{ij} be the number of vertices in the j th component just before the i th insertion. Thus $\mathbb{P}\{\exists i, j, M_{ij} > 2\gamma^{-1} \log n\} \leq \mathbb{P}\{M > 2\gamma^{-1} \log n\} = O(1/n)$. Since we have fewer than m time intervals of length ν during the first stage, and at most n connected components in \mathcal{G} , then the binomial tail inequality yields that

$$\begin{aligned} \mathbb{P}\{A\} &\leq \mathbb{P}\{A \cap [M \leq 2\gamma^{-1} \log n]\} + \mathbb{P}\{M > 2\gamma^{-1} \log n\} \\ &\leq mn \max_{i,j} \mathbb{P}\{A_{ij} \cap [M \leq 2\gamma^{-1} \log n]\} + O(1/n) \\ &\leq mn \max_{i,j} \mathbb{P}\{A_{ij} \cap [M_{ij} \leq 2\gamma^{-1} \log n]\} + O(1/n) \\ &\leq mn \max_{i,j} \mathbb{P}\{A_{ij} \mid [M_{ij} \leq 2\gamma^{-1} \log n]\} + O(1/n). \end{aligned}$$

The probability that the i th request hits the j th component is at most $2M_{ij}/n$, since there are two vertices involved in the request. The number of times that the j th component is hit in the subsequent $\nu - 1$ requests is distributed as $\text{Bin}(2\nu - 2, M_{ij}/n)$. Thus, using $\mathbb{P}\{\text{Bin}(n, p) \geq 2\} \leq (np)^2/2$, we have

$$\begin{aligned} \mathbb{P}\{A\} &\leq mn \left(\frac{4 \log n}{\gamma n} \right) \mathbb{P}\left\{ \text{Bin}\left(2\nu, \frac{4\gamma^{-1} \log n}{n}\right) \geq 2 \right\} + O\left(\frac{1}{n}\right) \\ &\leq mn \left(\frac{4 \log n}{\gamma n} \right) \left(\frac{8\nu \log n}{\gamma n} \right)^2 + O\left(\frac{1}{n}\right) \\ &= O\left(\frac{\log^3 n}{n^{1/2}}\right). \end{aligned}$$

The event A refers to the first stage only. Multiplying the bound by the number of stages $N = O(\log n)$, we can see that the probability that during some stage there is a connected component which is chosen more than twice in some interval of time of length ν goes to zero as $n \rightarrow \infty$.

Next, let $[a, b] = \{a, a + 1, \dots, b\}$, where $|b - a + 1| = \nu$ is a time interval of length ν , and let B be the event that $[a, b]$ is the first interval of time of length ν in which the queue \mathcal{Q} is never empty. Observe that if B is true, then the queue was empty at time $a - 1$. Let e_a, \dots, e_b be the edges associated with the ν requests appended to \mathcal{Q} during $[a, b]$. For $i \in [a, b]$, let T_i be the computational time needed for the algorithm to fully process the edge e_i , that is, the number of edges traversed or reversed during the whole process of serving the request, plus one if the edge is inserted. Let R_i be the number of vertices in the connected component to which the edge e_i belongs. Thus, $T_i \leq 4R_i$ for all $i \in [a, b]$. Using Chernoff's bounding method (see e.g., [19]), we see that for parameter $\lambda > 0$,

$$\begin{aligned} \mathbb{P}\{B\} &\leq \mathbb{P}\{T_a > \kappa, T_a + T_{a+1} > 2\kappa, \dots, T_a + \dots + T_b > \kappa\nu\} \\ &\leq \mathbb{P}\{T_a + \dots + T_b > \kappa\nu\} \\ &\leq \mathbb{P}\{R_a + \dots + R_b > \kappa\nu/4\} \\ &\leq e^{-\lambda\kappa\nu/4} \mathbf{E} \left[\exp \left(\sum_{i=a}^b \lambda R_i \right) \right]. \end{aligned}$$

Recall that $\mathcal{G}(m)^+$ denotes the union of $\mathcal{G}(m)$ and the edges that completed cycles. Suppose that we choose ν edges from the set of all m edges in the graph $\mathcal{G}(m)^+$ independently and uniformly without replacement. Let V_1, \dots, V_ν be the sizes of the components containing these edges. Notice that V_1, \dots, V_ν stochastically dominate R_a, \dots, R_b . On the other hand, suppose that we repeat the experiment of choosing ν edges from the set of all m edges in the graphs $\mathcal{G}(m)^+$ independently and uniformly but *with* replacement. Let V_1^*, \dots, V_ν^* be the values of these edges which are plainly independent and identically distributed. Hence, by Lemma 8, we see that

$$\begin{aligned} \mathbb{P}\{B\} &\leq e^{-\lambda\kappa\nu/4} \mathbf{E} \left[\exp \left(\sum_{i=1}^{\nu} \lambda V_i \right) \right] \\ &\leq e^{-\lambda\kappa\nu/4} \mathbf{E} \left[\exp \left(\sum_{i=1}^{\nu} \lambda V_i^* \right) \right] \\ &= e^{-\lambda\kappa\nu/4} \left(\mathbf{E} \left[e^{\lambda V_1^*} \right] \right)^\nu. \end{aligned}$$

By definition, V_1^* is distributed as the size of the component containing a uniformly chosen edge of $\mathbb{G}(n, m)$. In distribution, this is the same as the size of the component containing the *last* edge inserted into $\mathbb{G}(n, m)$, which is in turn stochastically dominated by $C(u) + C(v)$, where u and v are uniformly chosen vertices and $C(u)$ is the size of the component containing u in $\mathbb{G}(n, m - 1)$. Using Lemma 5 for $t \in [m]$,

$$\mathbb{P}\{V_1^* \geq t\} \leq \mathbb{P}\{C(u) + C(v) \geq t\} \leq 2\mathbb{P}\{C(u) \geq t/2\} \leq 2 \left(2e^{-\gamma(t/2-1)} \right),$$

where $\gamma = \frac{(1/2-\beta)^2}{2+4\beta} > 0$. Therefore,

$$\mathbf{E} \left[e^{\lambda V_1^*} \right] \leq 4 \sum_{t=1}^m e^{\lambda t} e^{-\gamma(t/2-1)} \leq 4e^{\lambda+\gamma/2} \sum_{t=0}^{\infty} e^{(\lambda-\gamma/2)t} \leq \frac{4e^{\lambda+\gamma/2}}{1 - e^{\lambda-\gamma/2}},$$

provided that $\lambda < \gamma/2$. Finally, we get

$$\mathbb{P}\{B\} \leq e^{-\lambda\kappa\nu/4} \left(\mathbf{E} \left[e^{\lambda V_1^*} \right] \right)^\nu \leq e^{-p\nu},$$

where $p := \lambda\kappa/4 - \lambda - \gamma/2 + \log(1 - e^{\lambda-\gamma/2}) - \log 4$, which is positive if we choose $\kappa > 4 + 4(-\gamma/2 - \log(1 - e^{\lambda-\gamma/2}) + \log 4) / \lambda$. For example, if we put $\lambda = \gamma/4$, then $\kappa = \lceil -40 \log(1 - e^{-\gamma}) / \gamma \rceil$ will suffice. With this choice, and since there are $N = \lceil s/m \rceil$ stages, and in each stage there are at most m intervals of time of length ν , we see that the probability that at some stage there is an interval of time of length ν in which the queue is never empty is at most $Nme^{-p\nu} = o(1)$. Indeed, this is true even if the interval is as short as $2 \log_p s = O(\log n)$. The proof now is complete. \square

Remarks.

1. The last step of the proof reveals that asymptotically almost surely the space consumed by the queue is indeed $O(\log n)$ because the queue is always empty at least once during any interval of $O(\log n)$ insertions.
2. Notice that as β approaches $1/2$, γ goes to zero, and hence, the constant κ increases to infinity.
3. As β increases to $1/2$, the worst-case search time of our algorithm is close to $4\alpha + 6$ w.h.p., where $\alpha := s/n$ is the load factor of the hash table. This performance can be beaten by other hashing algorithms when α is large enough. For example, when $\alpha \geq \log n / \log \log n$, the classical hashing with chaining, where only a single uniform hash function is utilized, achieves better performance than our algorithm: the maximum search time is known [27] to be at most $\alpha + \Theta(\sqrt{\alpha \log n})$ w.h.p. Also, when $\log_2 \log n \leq \alpha \leq (1/3) \log n / \log \log n$, the worst-case search time of greedy two-way chaining, which is $2 \log_2 \log n + 2\alpha + O(1)$ w.h.p., is the best known [4]. However, for $\alpha < \log_2 \log n$, our algorithm has the best worst-case search time among all known hashing with chaining algorithms that have constant worst-case insertion time.
4. We did not try to optimize the constant insertion time κ or the total space consumed by the algorithm. We believe that some aspects of the algorithm can be modified to improve its performance by a constant factor. For example, the use of the graph \mathcal{G} is probably unnecessary, and one can implement the operations on the graph directly on the hash table. It is also necessary to generalize the algorithm for the dynamic case, where deletions are allowed.

Appendix. In this appendix, we give the detailed pseudocode for the procedure `ConstantInsert`. Each element of the queue is the 5-tuple $[X, \text{Root1}, \text{Root2}, \text{IndexFrom}, \text{IndexTo}]$ defined as follows:

- `X` The pointer to the linked-list node that is currently in some chain of \mathcal{T} .
 - `Root1` Current computation of the root of vertex $f(x)$'s tree.
 - `Root2` Current computation of the root of vertex $g(x)$'s tree.
 - `IndexFrom` If `Root1` and `Root2` are roots, then the new arc should point from `IndexFrom`.
 - `IndexTo` If `Root1` and `Root2` are roots, then the new arc should point to `IndexTo`.
- The last two fields are computable from each other, so only four elements are actually needed. Additionally, recall that `X` points to a structure that contains the key, which we will refer to as `X.key`.

Pseudocode 6 ConstantInsert($x, \mathcal{G}, \mathcal{T}$)

```

1: Create new linked-list node, *X containing key x
2: Insert *X into  $\mathcal{T}[f(x)]$ 
3: Create new request,  $R_{\text{new}} = [X, f(x), g(x), f(x), g(x)]$ 
4: Append  $R_{\text{new}}$  to  $\mathcal{Q}$ 
5: Zero  $\lceil \frac{1}{\beta} \rceil$  more elements of  $\mathcal{G}'$ 
6:  $i \leftarrow 1$ 
7: repeat
8:    $i \leftarrow i + 1, R \leftarrow \mathcal{Q}.\text{Peek}()$ 
9:   if  $P[R.\text{Root1}] \neq \text{nil}$  then
10:     $R.\text{Root1} = P[R.\text{Root1}]$ 
11:   else if  $P[R.\text{Root2}] \neq \text{nil}$  then
12:     $R.\text{Root2} = P[R.\text{Root2}]$ 
13:   else if  $R.\text{Root1} = R.\text{Root2}$  then
14:     $\mathcal{Q}.\text{Pop}()$ 
15:   else
16:     $Y \leftarrow X[R.\text{IndexFrom}]$ 
17:    Link( $R.\text{IndexFrom}, R.\text{IndexTo}, R.X$ )
18:    if  $Y = \text{nil}$  then
19:       $\mathcal{Q}.\text{Pop}()$  {This will happen once for every new arc in  $\mathcal{G}$ .}
20:      if  $|\mathcal{G}| = \lfloor \beta n \rfloor$  then swap( $\mathcal{G}, \mathcal{G}'$ )
21:      else if  $f(Y.\text{key}) = R.\text{IndexFrom}$  then
22:         $R \leftarrow [Y, R.\text{Root1}, R.\text{Root2}, g(Y.\text{key}), f(Y.\text{key})]$ 
23:      else
24:         $R \leftarrow [Y, R.\text{Root1}, R.\text{Root2}, f(Y.\text{key}), g(Y.\text{key})]$ 
25:      end if
26:    end if
27: until  $i = \kappa$  OR  $\mathcal{Q} = \emptyset$ 

```

REFERENCES

- [1] N. ALON AND J. H. SPENCER, *The Probabilistic Method*, 2nd ed., John Wiley, New York, 2000.
- [2] D. ANGLUIN AND L. G. VALIANT, *Fast probabilistic algorithms for Hamiltonian paths and matchings*, J. Comput. System Sci., 18 (1979), pp. 155–193.
- [3] Y. AZAR, A. Z. BRODER, A. R. KARLIN, AND E. UPFAL, *Balanced allocations*, SIAM J. Comput., 29 (1999), pp. 180–200.
- [4] P. BERENBRINK, A. CZUMAJ, A. STEGER, AND B. VÖCKING, *Balanced allocations: The heavily loaded case*, in Proceedings of the 32nd Annual ACM Symposium on Theory of Computing (STOC), 2000, pp. 745–754.
- [5] A. BRODER AND M. MITZENMACHER, *Using multiple hash functions to improve IP lookups*, in Proceedings of the IEEE INFOCOM 2001 Conference (Anchorage, AK), 2001, pp. 1454–1463. Full version available as Technical report TR-03-00, Department of Computer Science, Harvard University, Cambridge, MA, 2000. Available online at <http://www.eecs.harvard.edu/~michaelm/NEWWORK/postscripts/iproute.pdf>
- [6] J. BYERS, J. CONSIDINE, AND M. MITZENMACHER, *Simple load balancing for distributed hash tables*, in Proceedings of the 2nd International Workshop on Peer-to-Peer Systems, 2003, pp. 80–87.
- [7] A. CZUMAJ, F. MEYER AUF DER HEIDE, AND V. STEMANN, *Contention resolution in hashing based shared memory simulations*, SIAM J. Comput., 29 (2000), pp. 1703–1739.
- [8] A. CZUMAJ AND V. STEMANN, *Randomized allocation processes*, Random Structures Algorithms, 18 (2001), pp. 297–331.

- [9] L. DEVROYE, *Branching processes and their applications in the analysis of tree structures and tree algorithms*, in Probabilistic Methods for Algorithmic Discrete Mathematics, M. Habib, C. McDiarmid, J. Ramirez-Alfonsin, and B. Reed, eds., Springer, Berlin, 1998, pp. 249–314.
- [10] L. DEVROYE AND P. MORIN, *Cuckoo hashing: Further analysis*, Inform. Process. Lett., 86 (2004), pp. 215–219.
- [11] M. DIETZFELBINGER, A. KARLIN, K. MEHLHORN, F. MEYER AUF DER HEIDE, H. ROHNERT, AND R. E. TARJAN, *Dynamic perfect hashing: Upper and lower bounds*, SIAM J. Comput., 23 (1994), pp. 738–761.
- [12] M. DIETZFELBINGER AND F. MEYER AUF DER HEIDE, *A new universal class of hash functions and dynamic hashing in real time*, in Proceedings of the 17th Annual International Colloquium on Automata, Languages and Programming, Lecture Notes in Comput. Sci. 443, Springer, New York, 1990, pp. 6–19.
- [13] M. DIETZFELBINGER AND F. MEYER AUF DER HEIDE, *High performance universal hashing, with applications to shared memory simulations*, in Data Structures and Efficient Algorithms, Lecture Notes in Comput. Sci. 594, Springer, Berlin, 1992, pp. 250–269.
- [14] P. ERDÖS, *Some remarks on the theory of graphs*, Bull. Amer. Math. Soc., 53 (1947), pp. 292–294.
- [15] P. ERDÖS AND A. RÉNYI, *On the evolution of random graphs*, Publ. Math. Ins. Hungar. Acad. Sci., 5 (1960), pp. 17–61.
- [16] M. FREDMAN, J. KOMLÓS, AND E. SZEMERÉDI, *Storing a sparse table with $O(1)$ worst case access time*, J. ACM, 31 (1984), pp. 538–544.
- [17] H. GAJEWSKA AND R. E. TARJAN, *Dequeues with heap order*, Inform. Process. Lett., 22 (1986), pp. 197–200.
- [18] G. H. GONNET, *Expected length of the longest probe sequence in hash code searching*, J. ACM, 28 (1981), pp. 289–304.
- [19] W. HOEFFDING, *Probability inequalities for sums of bounded random variables*, J. Amer. Statist. Assoc., 58 (1963), pp. 13–30.
- [20] S. JANSON, D. E. KNUTH, T. ŁUCZAK, AND B. PITTEL, *The birth of the giant component*, Random Structures Algorithms, 4 (1993), pp. 233–358.
- [21] S. JANSON, T. ŁUCZAK, AND A. RUCIŃSKI, *Random Graphs*, John Wiley, New York, 2000.
- [22] R. M. KARP, *The transitive closure of a random digraph*, Random Structures Algorithms, 1 (1990), pp. 73–93.
- [23] E. MALALLA, *Two-Way Hashing with Separate Chaining and Linear Probing*, Ph.D. thesis, School of Computer Science, McGill University, Montreal, Quebec, Canada, 2004.
- [24] M. MITZENMACHER, A. RICHA, AND R. SITARAMAN, *The power of two random choices: A survey of techniques and results*, in Handbook of Randomized Computing, P. Pardalos, S. Rajasekaran, and J. Rolim, eds., Kluwer, Dordrecht, 2001, pp. 255–312.
- [25] R. PAGH, *On the cell probe complexity of membership and perfect hashing*, in Proceedings of the 33rd Annual ACM Symposium on Theory of Computing (STOC), 2001, pp. 425–432.
- [26] R. PAGH AND F. F. RODLER, *Cuckoo hashing*, J. Algorithms, 51 (2004), pp. 122–144.
- [27] M. RAAB AND A. STEGER, *Balls into bins—a simple and tight analysis*, in Proceedings of the 2nd Annual Workshop on Randomization and Approximation Techniques in Computer Science, Lecture Notes in Comput. Sci. 1518, Springer, Berlin, 1998, pp. 159–170.
- [28] B. VÖCKING, *How asymmetry helps load balancing*, in Proceedings of the 40th Annual IEEE Symposium on Foundations of Computer Science (FOCS), 1999, pp. 131–141.

CACHE-OBLIVIOUS B-TREES*

MICHAEL A. BENDER[†], ERIK D. DEMAINÉ[‡], AND MARTIN FARACH-COLTON[§]

Abstract. This paper presents two dynamic search trees attaining near-optimal performance on any hierarchical memory. The data structures are independent of the parameters of the memory hierarchy, e.g., the number of memory levels, the block-transfer size at each level, and the relative speeds of memory levels. The performance is analyzed in terms of the number of memory transfers between two memory levels with an arbitrary block-transfer size of B ; this analysis can then be applied to every adjacent pair of levels in a multilevel memory hierarchy. Both search trees match the optimal search bound of $\Theta(1 + \log_{B+1} N)$ memory transfers. This bound is also achieved by the classic B-tree data structure on a two-level memory hierarchy with a known block-transfer size B . The first search tree supports insertions and deletions in $\Theta(1 + \log_{B+1} N)$ amortized memory transfers, which matches the B-tree's worst-case bounds. The second search tree supports scanning S consecutive elements optimally in $\Theta(1 + S/B)$ memory transfers and supports insertions and deletions in $\Theta(1 + \log_{B+1} N + \frac{\log^2 N}{B})$ amortized memory transfers, matching the performance of the B-tree for $B = \Omega(\log N \log \log N)$.

Key words. memory hierarchy, cache efficiency, data structures, search trees

AMS subject classifications. 68P05, 68P30, 68P20

DOI. 10.1137/S0097539701389956

1. Introduction. The memory hierarchies of modern computers are becoming increasingly steep. Typically, an L1 cache access is two orders of magnitude faster than a main memory access and six orders of magnitude faster than a disk access [27]. Thus, it is dangerously inaccurate to design algorithms assuming a flat memory with uniform access times.

Many computational models attempt to capture the effects of the memory hierarchy on the running times of algorithms. There is a tradeoff between the accuracy of the model and its ease of use. One body of work explores multilevel memory hierarchies [2, 3, 5, 7, 43, 44, 49, 51], though the proliferation of parameters in these models makes them cumbersome for algorithm design. A second body of work concentrates on two-level memory hierarchies, either main memory and disk [4, 12, 32, 49, 50] or cache and main memory [36, 45]. With these models the programmer must anticipate which level of the memory hierarchy is the bottleneck. For example, a B-tree that has been tuned to run on disk has poor performance in memory.

1.1. Cache-oblivious algorithms. The cache-oblivious model enables us to reason about a simple two-level memory but prove results about an unknown multilevel memory. This model was introduced by Frigo et al. [31] and Prokop [40].

*Received by the editors May 31, 2001; accepted for publication (in revised form) May 25, 2005; published electronically October 7, 2005. A preliminary version of this paper appeared in FOCS 2000 [18].

<http://www.siam.org/journals/sicomp/35-2/38995.html>

[†]Department of Computer Science, State University of New York, Stony Brook, NY 11794-4400 (bender@cs.sunysb.edu). This author's work was supported in part by HRL Laboratories, ISX Corporation, Sandia National Laboratories, and NSF grants EIA-0112849 and CCR-0208670.

[‡]Computer Science and Artificial Intelligence Laboratory, MIT, 32 Vassar Street, Cambridge, MA 02139 (edemaine@mit.edu). This author's work was supported in part by NSF grant EIA-0112849.

[§]Department of Computer Science, Rutgers University, Piscataway, NJ 08855 (farach@cs.rutgers.edu). This author's work was supported by NSF grant CCR-9820879.

They show that several basic problems—namely, matrix multiplication, matrix transpose, the fast Fourier transform (FFT), and sorting—have optimal algorithms that are cache oblivious. Optimal cache-oblivious algorithms have also been found for LU decomposition [21, 46] and static binary search [40]. These algorithms perform an asymptotically optimal number of memory transfers for *any* memory hierarchy and at *all levels* of the hierarchy. More precisely, the number of memory transfers between any two levels is within a constant factor of optimal. In particular, any linear combination of the transfer counts is optimized.

The theory of cache-oblivious algorithms is based on the *ideal-cache model* of Frigo et al. [31] and Prokop [40]. In the ideal-cache model there are two levels in the memory hierarchy, called *cache* and *main memory*, although they could represent any pair of levels. Main memory is partitioned into *memory blocks*, each consisting of a fixed number B of consecutive cells. The cache has size M , and consequently has capacity to store M/B memory blocks.¹ In this paper, we require that M/B be greater than a sufficiently large constant. The cache is *fully associative*, that is, it can contain an arbitrary set of M/B memory blocks at any time.

The parameters B and M are unknown to the cache-oblivious algorithm or data structure. As a result, the algorithm cannot explicitly manage memory, and this burden is taken on by the system. When the algorithm accesses a location in memory that is not stored in cache, the system fetches the relevant memory block from main memory in what is called a *memory transfer*. If the cache is full, a memory block is elected for replacement based on an optimal offline analysis of the future memory accesses of the algorithm.

Although this model may superficially seem unrealistic, Frigo et al. show that it can be simulated by essentially any memory system with a small constant-factor overhead. Thus, if we run a cache-oblivious algorithm on a multilevel memory hierarchy, we can use the ideal-cache model to analyze the number of memory transfers between each pair of adjacent levels. See [31, 40] for details.

The concept of algorithms that are uniformly optimal across multiple memory models was considered previously by Aggarwal et al. [2]. These authors introduce the hierarchical memory model (HMM) model, in which the cost to access memory location x is $\lceil f(x) \rceil$, where $f(x)$ is monotone nondecreasing and polynomially bounded. They give algorithms for matrix multiplication and the FFT that are optimal for any cost function $f(x)$. One distinction between the HMM model and the cache-oblivious model is that, in the HMM model, memory is managed by the algorithm designer, whereas in the cache-oblivious model, memory is managed by the existing caching and paging mechanisms. Also, the HMM model does not include block transfers, though Aggarwal, Chandra, and Snir [3] later extended the HMM to the block transfer (BT) model to take into account block transfers. In the BT model the algorithm can choose and vary the block size, whereas in the cache-oblivious model the block size is fixed and unknown.

1.2. B-trees. In this paper, we initiate the study of dynamic cache-oblivious data structures by developing cache-oblivious search trees.

The classic I/O-efficient search tree is the *B-tree* [13]. The basic idea is to maintain a balanced tree of N elements with node fanout proportional to the memory block size B . Thus, one block read determines the next node out of $\Theta(B)$ nodes, so a search

¹Note that B and M are parameters, not constants. Consequently, they must be preserved in asymptotic notation in order to obtain accurate running-time estimates.

TABLE 1

Related work in cache-oblivious data structures. These results, except the static search tree of [40], appeared after the conference version [18] of this paper.

B-tree	<ul style="list-style-type: none"> • Simplification via packed-memory structure/low-height trees [20, 25] • Simplification and persistence via exponential structures [42, 17] • Implicit [29, 30]
Static search trees	<ul style="list-style-type: none"> • Basic layout [40] • Experiments [35] • Optimal constant factor [14]
Linked lists supporting scans	[15]
Priority queues	[8, 23, 26]
Trie layout	[6, 19]
Computational geometry	<ul style="list-style-type: none"> • Distribution sweeping [22] • Voronoi diagrams [34] • Orthogonal range searching [1, 9] • Rectangle stabbing [10]
Lower bounds	[24]

completes in $\Theta(1 + \log_{B+1} N)$ memory transfers.² A simple information-theoretic argument shows that this bound is optimal.

The B-tree is designed for a two-level hierarchy, and the situation becomes more complex with more than two levels. We need a multilevel structure, with one level per transfer block size. Suppose $B_1 > B_2 > \dots > B_L$ are the block sizes between the $L + 1$ levels of memory. At the top level we have a B_1 -tree; each node of this B_1 -tree is a B_2 -tree; etc. Even when it is possible to determine all these parameters, such a data structure is cumbersome. Also, each level of recursion incurs a constant-factor wastage in storage, in order to amortize dynamic changes, leading to suboptimal memory-transfer performance for $L = \omega(1)$.

1.3. Results. We develop two cache-oblivious search trees. These results are the first demonstration that even irregular and dynamic problems, such as data structures, can be solved efficiently in the cache-oblivious model. Since the conference version [18] of this paper appeared, many other data-structural problems have been addressed in the cache-oblivious model; see Table 1. Our results achieve the memory-transfer bounds listed below. The parameter N denotes the number of elements stored in the tree. *Updates* refer to both key insertions and deletions.

1. The first cache-oblivious search tree attains the following memory-transfer bounds:
 - Search:* $O(1 + \log_{B+1} N)$, which is optimal and matches the search bound of B-trees.
 - Update:* $O(1 + \log_{B+1} N)$ amortized, which matches the update bound of B-trees, though the B-tree bound is worst case.
2. The second cache-oblivious search tree adds the *scan* operation (also called the *range search* operation). Given a key x and a positive integer S , the scan operation accesses S elements in key order, starting after x . The memory-transfer bounds are as follows:
 - Search:* $O(1 + \log_{B+1} N)$.

²We use $B + 1$ as the base of the logarithm to correctly capture that the special case of $B = 1$ corresponds to the RAM.

Scan: $O(1 + S/B)$, which is optimal.

Update: $O(1 + \log_{B+1} N + \frac{\log^2 N}{B})$ amortized, which matches the B-tree update bound of $O(1 + \log_{B+1} N)$ when $B = \Omega(\log N \log \log N)$.

This last relation between B and N usually holds in external memory but often does not hold in internal memory.

In the development of these data structures, we build and identify tools for cache-oblivious manipulation of data. These tools have since been used in many of the cache-oblivious data structures listed in Table 1. In section 2.1, we show how to linearize a tree according to what we call the *van Emde Boas layout*, along the lines of Prokop’s static search tree [40]. In section 2.2, we describe a type of *strongly weight-balanced search tree* [11] useful for maintaining locality of reference. Following the work of Itai, Konheim, and Rodeh [33] and Willard [52, 53, 54], we develop a *packed-memory array* for maintaining an ordered collection of N items in an array of size $O(N)$ subject to insertions and deletions in $O(1 + \frac{\log^2 N}{B})$ amortized memory transfers; see section 2.3. This structure can be thought of as a cache-oblivious *linked list* that supports scanning S consecutive elements in $O(1 + S/B)$ memory transfers (instead of the naïve $O(S)$) and updates in $O(1 + \frac{\log^2 N}{B})$ amortized memory transfers.

1.4. Notation. We define the *hyperfloor* of x , denoted $\lfloor\!\lfloor x \rfloor\!\rfloor$, to be $2^{\lfloor \log x \rfloor}$, i.e., the largest power of 2 smaller than x .³ Thus, $x/2 < \lfloor\!\lfloor x \rfloor\!\rfloor \leq x$. Similarly, the *hyperceiling* $\lceil\!\lceil x \rceil\!\rceil$ is defined to be $2^{\lceil \log x \rceil}$. Analogously, we define *hyperhyperfloor* and *hyperhyperceiling* by $\lfloor\!\!\lfloor x \!\!\rfloor\!\rfloor = 2^{\lfloor \log x \rfloor}$ and $\lceil\!\!\lceil x \!\!\rceil\!\rceil = 2^{\lceil \log x \rceil}$. These operators satisfy $\sqrt{x} < \lfloor\!\!\lfloor x \!\!\rfloor\!\rfloor \leq x$ and $x \leq \lceil\!\!\lceil x \!\!\rceil\!\rceil < x^2$.

2. Tools for cache-oblivious data structures.

2.1. Static layout and searches. We first present a cache-oblivious *static* search-tree structure, which is the starting point for the dynamic structures. Consider a $O(\log N)$ -height search tree in which every node has at least two and at most a constant number of children and in which all leaves are on the same level. We describe a mapping from the nodes of the tree to positions in memory. The cost of any search in this layout is $\Theta(1 + \log_{B+1} N)$ memory transfers, which is optimal up to constant factors. Our layout is a modified version of Prokop’s layout for a complete binary tree whose height is a power of 2 [40, pp. 61–62]. We call the layout the *van Emde Boas layout* because it resembles the van Emde Boas data structure [47, 48].⁴

The van Emde Boas layout proceeds recursively. Let h be the height of the tree, or more precisely, the number of levels of nodes in the tree. Suppose first that h is a power of 2. Conceptually split the tree at the middle level of edges, between nodes of height $h/2$ and $h/2 + 1$. This breaks the tree into the *top recursive subtree* A of height $h/2$ and several *bottom recursive subtrees* B_1, B_2, \dots, B_ℓ , each of height $h/2$. If all nonleaf nodes have the same number of children, then the recursive subtrees all have size roughly \sqrt{N} , and ℓ is roughly \sqrt{N} . The layout of the tree is obtained by recursively laying out each subtree and combining these layouts in the order $A, B_1, B_2, \dots, B_\ell$; see Figure 1.

If h is not a power of 2, we assign a number of levels that is a power of 2 to the bottom recursive subtrees and assign the remaining levels to the top recursive subtree. More precisely, the bottom subtrees have height $\lceil\!\lceil h/2 \rceil\!\rceil$ ($= \lfloor\!\lfloor h - 1 \rfloor\!\rfloor$) and

³All logarithms are base 2 if not otherwise specified.

⁴We do not use a van Emde Boas tree—we use a normal tree with pointers from each node to its parent and children—but the order of the nodes in memory is reminiscent of van Emde Boas trees.

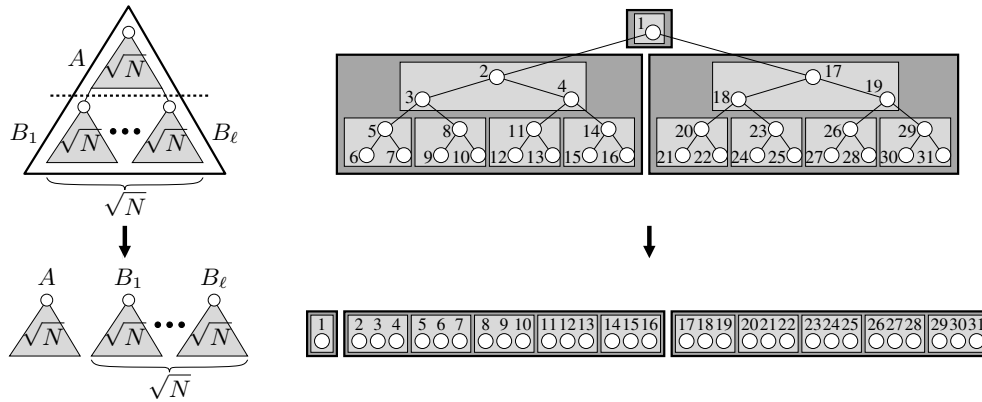


FIG. 1. The van Emde Boas layout. Left: in general; right: of a tree of height 5.

the top subtree has height $h - \lceil h/2 \rceil$. This rounding scheme is important for later dynamic structures because the heights of the cut lines in the lower trees do not vary with N . In contrast, this property is not shared by the simple rounding scheme of assigning $\lfloor h/2 \rfloor$ levels to the top recursive subtree and $\lceil h/2 \rceil$ levels to the bottom recursive subtrees.

The memory-transfer analysis views the van Emde Boas layout at a particular *level of detail*. Each level of detail is a partition of the tree into disjoint recursive subtrees. In the finest level of detail, 0, each node forms its own recursive subtree. In the coarsest level of detail, $\lceil \log_2 h \rceil$, the entire tree forms the unique recursive subtree. Level of detail k is derived by starting with the entire tree, recursively partitioning it as described above, and exiting a branch of the recursion upon reaching a recursive subtree of height $\leq 2^k$. The key property of the van Emde Boas layout is that, at any level of detail, each recursive subtree is stored in a contiguous block of memory.

One useful consequence of our rounding scheme is the following.

LEMMA 1. *At level of detail k all recursive subtrees except the one containing the root have the same height of 2^k . The recursive subtree containing the root has height between 1 and 2^k inclusive.*

Proof. The proof follows from a simple induction on the level of detail. Consider a tree T of height h . At the coarsest level of detail, $\lceil \log_2 h \rceil$, there is a single recursive subtree, which includes the root. In this case the lemma is trivial. Suppose by induction that the lemma holds for level of detail k . In this level of detail the recursive subtree containing the root of T has height h' , where $1 \leq h' \leq 2^k$, and all other recursive subtrees have height 2^k . To progress to the next finer level of detail, $k - 1$, all recursive subtrees that do not contain the root are recursively split once more so that they have height 2^{k-1} . If the height h' of the top recursive subtree is at most 2^{k-1} , then it is not split in level of detail $k - 1$. Otherwise, the root is split into bottom recursive subtrees of height 2^{k-1} and a top recursive subtree of height $h'' \leq 2^{k-1}$. The inductive step follows. \square

LEMMA 2. *Consider an N -node search tree T that is stored in a van Emde Boas layout. Suppose that each node in T has between $\delta \geq 2$ and $\Delta = O(1)$ children. Let h be the height of T . Then a search in T uses at most $4 \lceil \log_\delta \Delta \log_{B+1} N + \log_{B+1} \Delta \rceil = O(1 + \log_{B+1} N)$ memory transfers.*

Proof. Let k be the coarsest level of detail such that every recursive subtree contains at most B nodes; see Figure 2. Thus, every recursive subtree is stored in at

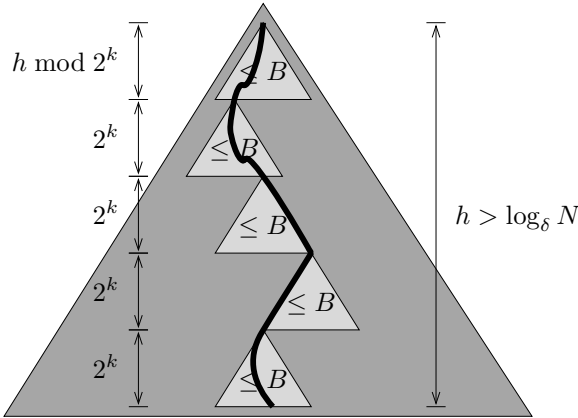


FIG. 2. The recursive subtrees visited by a root-to-leaf search path in level of detail k .

most two memory blocks. Because tree T has height h , $\lceil h/2^k \rceil$ recursive subtrees are traversed in each search, and thus at most $2\lceil h/2^k \rceil$ memory blocks are transferred. Because the tree has height h , $\delta^{h-1} < N < \Delta^h$, that is, $\log_\Delta N < h < \log_\delta N + 1$. Because a tree of height 2^{k+1} has more than B nodes, $\Delta^{2^{k+1}} > B$, so $\Delta^{2^{k+1}} \geq B + 1$. Thus, $2^k \geq \frac{1}{2} \log_\Delta(B + 1)$. Therefore, the maximum number of memory transfers is

$$\begin{aligned} 2 \left\lceil \frac{h}{2^k} \right\rceil &\leq 4 \left\lceil \frac{1 + \log_\delta N}{\log_\Delta(B + 1)} \right\rceil = 4 \left\lceil \left(1 + \frac{\log N}{\log \delta}\right) \left(\frac{\log \Delta}{\log(B + 1)}\right) \right\rceil \\ &= 4 \lceil \log_\delta \Delta \log_{B+1} N + \log_{B+1} \Delta \rceil. \end{aligned}$$

Because δ and Δ are constants, this bound is $O(1 + \log_{B+1} N)$. \square

2.2. Strongly weight-balanced search trees. To convert the static layout into a dynamic layout, we use a dynamic balanced search tree. We require the following two properties of the balanced search tree.

PROPERTY 1 (descendant amortization). *Suppose that whenever we rebalance a node v (i.e., modify it to keep balance) we also touch all of v 's descendants. Then the amortized number of elements touched per insertion is $O(\log N)$.*

PROPERTY 2 (strong weight balance). *For some constant d , every node v at height h has $\Theta(d^h)$ descendants.*

Property 1 is normally implied by Property 2 as well as by a weaker property called weight balance. A tree is *weight balanced* if, for every node v , its left subtree (including v) and its right subtree (including v) have sizes that differ by at most a constant factor. Weight balancedness guarantees a relative bound between subtrees with a common root, so the size difference between subtrees of the same height may be large. In contrast, strong weight balance requires an absolute constraint that relates the sizes of all subtrees at the same level. For example, $\text{BB}[\alpha]$ trees [38] are weight-balanced binary search trees based on rotations, but they are not strongly weight balanced.

Search trees that satisfy Properties 1 and 2 include weight-balanced B-trees [11], deterministic skip lists [37], and skip lists [41] in the expected sense. We choose to use weight-balanced B-trees defined as follows.

DEFINITION 3 (weight-balanced B-tree [11]). *A rooted tree T is a weight-balanced*

B-tree with branching parameter d , where $d > 4$, if the following conditions hold:⁵

1. All leaves of T have the same depth.
2. The root of T has more than one child.
3. Balance: Consider a nonroot node u at height h in the tree. (Leaves have height 1.) The weight $w(u)$ of u is the number of nodes in the subtree rooted at u . This weight is bounded by

$$\frac{d^{h-1}}{2} \leq w(u) \leq 2d^{h-1}.$$

4. Amortization: If a nonroot node u at height h is rebalanced, then $\Omega(d^h)$ updates are necessary before u is rebalanced again. That is, $w(u) - d^{h-1}/2 = \Theta(d^h)$ and $2d^{h-1} - w(u) = \Theta(d^h)$.⁶

Conditions 1–4 have the following consequence:

5. The root has between 2 and $4d$ children. All internal nodes have between $d/4$ and $4d$ children. The height of the tree is $O(1 + \log_d N)$.

From the strong weight balancedness of the subtree rooted at a node, we can conclude strong weight balancedness of the top a levels in such a subtree, as follows.

LEMMA 4. Consider the subtree A of a weight-balanced B-tree containing a node v , its children, its grandchildren, etc., for exactly a levels. Then $|A| < 4d^a$.

Proof. Let T denote the subtree rooted at a node of height h . Consider the descendants of v down precisely a levels, and let B_1, B_2, \dots, B_k be the subtrees rooted at those nodes. In other words, the B_i 's are the children subtrees of A . Let $b = h - a$ denote the height of each B_i . Because T and the B_i 's are strongly weight balanced, $|T| \leq 2d^{h-1}$ and $|B_i| \geq \frac{1}{2}d^{b-1}$ for all i . Now $|B_1| + \dots + |B_k| < |T| \leq 2d^{h-1}$, so $k \leq 4d^{h-b} = 4d^a$. But the number of nodes in A is less than the number of children subtrees B_1, B_2, \dots, B_k , so the result follows. \square

Next we show how to perform updates, making a small modification to the presentation in [11]. Specifically, [11] performs deletions using the global rebalancing technique of [39], where deleted nodes are treated as “ghost” nodes to be removed when the tree is periodically reassembled. In the cache-oblivious model, we need to service deletions immediately to avoid large holes in the structure.

Insertions. We search down the tree to find where to insert a new leaf w . After inserting w , some ancestors of w may become unbalanced. That is, some ancestor node u at height h may have weight greater than $2d^{h-1}$. We bring the ancestors of w into balance starting from the ancestors closest to the leaves. If a node u at height h is out of balance, then we split u into two nodes u_1 and u_2 , which share the node u 's children, v_1, \dots, v_k . We can divide the children fairly evenly as follows. Find the longest sequence of $v_1, \dots, v_{k'}$ such that their total weight is at most $\lceil w(u)/2 \rceil$, that is, $\sum_{i=1}^{k'} w(v_i) \leq \lceil w(u)/2 \rceil$. Thus, $\lceil w(u)/2 \rceil - 2d^{h-2} + 1 \leq w(u_1) \leq \lceil w(u)/2 \rceil$ and $\lfloor w(u)/2 \rfloor \leq w(u_2) \leq \lfloor w(u)/2 \rfloor + 2d^{h-2} - 1$. Because $d > 4$, we continue to satisfy the properties of Definition 3. In particular, at least $\Theta(d^h)$ insertions or deletions are needed before either u_1 or u_2 is split.

Deletions. Deletions are similar to insertions. As before, we search down the tree to find which leaf w to delete. After deleting w , some ancestors of w may become unbalanced. That is, some ancestor node u at height h may have weight lower than $\frac{1}{2}d^{h-1}$. We merge u with one of its neighbors. After merging u , it might now have

⁵In [11] there is also a leaf parameter $k > 0$, but we simply fix $k = 1$.

⁶This property is not included in the definition in [11], but it is an important invariant satisfied by the structure.

a weight larger than its upper bound, so we immediately split it into two nodes as described in the insertion algorithm. A slightly more subtle problem is that the newly merged node may be just below the upper threshold and may need splitting soon thereafter. We can handle this problem in several ways. Here, we split a merged node v if it has weight greater than $\frac{7}{4}d^{h-1}$, thus producing two nodes of weight at least $\frac{7}{8}d^{h-1} - d^{h-2} \geq \frac{5}{8}d^{h-1}$ and at most $\frac{9}{8}d^{h-1}$. Thus, condition 4 is guaranteed. As with insertions, we handle such out-of-bound nodes u in order of increasing height.

2.3. Packed-memory array. A *packed-memory array* maintains N elements in order in an array of size $P = cN$ subject to element deletion and element insertion between two existing elements. The remaining fraction $1 - c$ of the array is blank. The packed-memory array must achieve two seemingly contradictory goals. On the one hand, we should pack the nodes densely so that scanning is fast: S consecutive elements must occupy $O(S)$ cells in the array so that scanning those elements uses $O(1 + S/B)$ memory transfers. On the other hand, we should leave enough blank space between the nodes to permit future insertions to be handled quickly. Meeting these two goals makes packed-memory arrays useful for storing dynamic linear data cache obliviously. We achieve the following balance.

THEOREM 5. *For any desired $c > 1$, the packed-memory array maintains N elements in an array of size cN and supports insertions and deletions in $O(1 + \frac{\log^2 N}{B})$ amortized memory transfers and scanning S consecutive elements in $O(1 + S/B)$ memory transfers.*

Our data structure and analysis closely follow Itai, Konheim, and Rodeh [33]. They consider the same problem of maintaining elements in order in an array of linear size, but in a different cost model and without the scanning requirement. Their structure moves $O(\log^2 N)$ amortized elements per insertion, but has no guarantee on the number of memory transfers for inserting, deleting, or scanning. This structure has been deamortized by Willard [52, 53, 54] and subsequently simplified by Bender et al. [16].⁷

At a high level, the packed-memory array keeps every interval of the array of size $\Omega(1)$ a constant fraction full, where the constant fraction depends on the interval size. When an interval of the array becomes too full or too empty, we evenly spread out (*rebalance*) the elements within a larger interval. It remains to specify the size of the interval to rebalance and the thresholds determining when an interval is too full or too empty.

Tree structure and thresholds. We divide the array into segments, each of size $\Theta(\lg P)$, so that the number of segments is a power of 2. We then implicitly build a perfect binary tree on top of these $\Theta(P / \lg P)$ segments, making each segment a leaf. Each node in the tree represents the subarray containing the segments in the subtree rooted at this node. In particular, the root node represents the entire array, and each leaf node represents a single segment.

To define the thresholds controlling rebalance, we need additional terminology. The *capacity* of a node u in the tree, denoted $\text{capacity}(u)$, is the size of its subarray. The *density* of a node u in the tree, denoted $\text{density}(u)$, is the number of elements stored in u 's subarray divided by $\text{capacity}(u)$. Define the root node to have depth 0 and the leaf nodes to have depth $d = \lg \Theta(P / \lg P)$.

For each node, we define two *density thresholds* specifying the desired range on

⁷This problem is closely related to, but distinct from, the problem of answering linked-list order queries [28, 16].

the node's density. Let $0 < \rho_d < \rho_0 < \tau_0 < \tau_d = 1$ be arbitrary constants. For a node u at depth k , the *upper-bound density threshold* τ_k is $\tau_0 + \frac{\tau_d - \tau_0}{d}k$ and the *lower-bound density threshold* ρ_k is $\rho_0 - \frac{\rho_0 - \rho_d}{d}k$. Thus,

$$0 < \rho_d < \rho_{d-1} < \dots < \rho_0 < \tau_0 < \tau_1 < \dots < \tau_d = 1.$$

A node u at depth k is *within threshold* if $\rho_k \leq \text{density}(u) \leq \tau_k$.

The main difference between our packed-memory array and the data structure of [33] is that we add lower-bound thresholds.

Insertion and deletion. To insert an element x , we proceed as follows. First we find the leaf node w where x belongs. If leaf w has free space, then we *rebalance* w by evenly distributing x and the $\Theta(\lg P)$ elements in the leaf. Otherwise, we proceed up the tree until we find the first ancestor u of w that is within threshold. Then we *rebalance* node u by evenly distributing all elements in u 's subarray throughout that subarray. Leaf w is now within threshold and we can insert x as before.

Deleting an element is similar. To delete an element x , we remove x from the leaf node w containing x . If w is still within threshold, we are done. Otherwise, we find the lowest ancestor u of w that is within threshold and we rebalance u . Leaf w is now within threshold.

Although we describe these algorithms conceptually as visiting nodes in a tree, the tree is not actually stored. The operations are implemented by two parallel scans, one moving left and one moving right, that visit the subarrays specified implicitly by the tree nodes along a leaf-to-root path. During these scans we count the number of elements found and the capacity traversed, stopping when the density of a subarray is within threshold. The memory-transfer cost of the scans is $O(1 + K/B)$, where K is the capacity of the subarray traversed. We then rebalance this subarray using $O(1)$ additional scans, so the total cost is $O(1 + K/B)$; K is also the capacity of the subarray rebalanced.

Whenever N changes by a constant factor, we rebuild the structure by copying the data into another array with newly computed densities.

Analysis. Next we bound the amortized size of a rebalance during an insertion or deletion. Suppose that we rebalance a node u at depth k . The rebalance was triggered by an insertion or deletion in the subarray of some child v of u . Before rebalancing, u is within threshold, i.e., $\rho_k \leq \text{density}(u) \leq \tau_k$, and v is not within threshold, i.e., $\text{density}(v) > \tau_{k+1}$ or $\text{density}(v) < \rho_{k+1}$. After rebalancing, the density of v (and of v 's sibling) is not only within threshold, but also within the density thresholds of its parent u , i.e., $\rho_k \leq \text{density}(v) \leq \tau_k$. Before the density of node v (or its sibling) next exceeds its upper-bound threshold, we must have at least $(\tau_{k+1} - \tau_k)$ capacity(v) additional insertions within its subarray. Similarly, before the density of node v (or its sibling) next falls below its lower-bound threshold, we must have at least $(\rho_k - \rho_{k+1})$ capacity(v) additional deletions within its subarray.

We charge the size capacity(u) of the rebalance at u to its child v . Therefore the amortized size of a rebalance per insertion into v 's subarray is

$$\frac{\text{capacity}(u)}{\text{capacity}(v)(\tau_{k+1} - \tau_k)} = \frac{2}{\tau_{k+1} - \tau_k} = \frac{2d}{\tau_d - \tau_0} = O(\lg P),$$

and the amortized size of a rebalance per deletion in v 's subarray is

$$\frac{\text{capacity}(u)}{\text{capacity}(v)(\rho_k - \rho_{k+1})} = \frac{2}{\rho_k - \rho_{k+1}} = \frac{2d}{\rho_0 - \rho_d} = O(\lg P).$$

When we insert or delete an element x , we insert or delete within $d = O(\lg P)$ different subintervals containing x . Therefore the total amortized size of a rebalance per insertion or deletion is $O(\lg^2 P) = O(\lg^2 N)$.

Because the amortized size of a rebalance per insertion or deletion is $O(\lg^2 N)$, the amortized number of memory transfers per insertion or deletion is $O(1 + \frac{\lg^2 N}{B})$. This concludes the proof of Theorem 5.

3. Main structure. We begin by describing a simple approach to cache-oblivious B-trees that is not efficient by itself and then describe the modifications and improvements necessary to obtain our structure.

The simple approach uses the principle of indirection to support fast updates. We partition the N elements into consecutive groups of $\Theta(\log N)$ elements each, and store each group in a separate array of size $\Theta(\log N)$. We maintain a standard balanced binary search tree, such as an AVL tree, on the minimum element from each group. This *top tree* thus stores $\Theta(N / \lg N)$ elements. Each element in the top tree stores a pointer to the corresponding group array, and vice versa. Most insertions and deletions can simply rewrite one of the group arrays, which costs $O(1 + \frac{\lg N}{B})$ memory transfers. Whenever a group grows by a constant factor (e.g., to $2 \lg N$) or shrinks by a constant factor (e.g., to $\frac{1}{2} \lg N$), we merge and/or split groups at a cost of $O(1 + \frac{\lg N}{B})$ memory transfers and then perform the corresponding deletion and/or insertion in the top tree at a cost of $O(\log N)$ memory transfers. The latter cost can be charged to the $\Omega(\log N)$ updates that caused the overflow or underflow, for an amortized $O(1)$ cost. Therefore the total amortized update cost is $O(1 + \frac{\lg N}{B}) = O(\log_{B+1} N)$ memory transfers.

The problem with this structure is that searching in the balanced binary search tree costs $O(\log N)$ memory transfers. The moral is that we can afford to have relatively slow insertions and deletions in the upper tree, but we need to have fast searches.

The next layer of complexity is to keep the top tree in an (approximate) van Emde Boas layout, which brings the total search time down to $O(\log_{B+1} N)$ memory transfers. (The cost to scan a group array is $O(1 + \frac{\log N}{B})$, which is negligible.) However, it is not clear how to preserve this van Emde Boas layout of a search tree under insertions and deletions. First we show that, if we use a weight-balanced B-tree from section 2.2 for the top tree, there are few changes to the relative order of elements in the van Emde Boas layout, at least in the amortized sense. Nonetheless, when we insert into the middle of the tree, we need to have extra space for newly created nodes. If we keep the tree layout in an array, each insertion may require shifting most of the tree, even though the relative order of elements changes very little. Instead we use a packed-memory array from section 2.3 to store the van Emde Boas layout, allowing us to make room for changes in the tree.

An additional technical complication arises in maintaining pointers to nodes that move during an update. We search in the top tree by following pointers from nodes to their children, represented by indices into the packed-memory array. When we insert or delete an element in the packed-memory array, amortized $O(\log^2 N)$ elements move. Any element that is moved must let its parent know where it has gone. Thus, each node must have a pointer to its parent, and so each node must also let its children know where it has moved. The $O(\log^2 N)$ nodes that move can have $O(\log^2 N)$ children scattered throughout the packed-memory array, each in separate memory blocks. Thus an insertion or deletion can potentially induce $O(\log^2 N)$ memory transfers to update these disparately located pointers. We can afford this large

cost in an amortized sense by adding another level of $\Theta(\log N)$ indirection.

The overall structure of our cache-oblivious B-tree therefore has three levels. The top level is a weight-balanced B-tree on $\Theta(N / \log^2 N)$ elements stored according to a van Emde Boas layout in a packed-memory array. The middle level is a collection of $\Theta(N / \log^2 N)$ groups of $\Theta(\log N)$ elements each. The bottom level is a collection of $\Theta(N / \log N)$ groups of $\Theta(\log N)$ elements each.

In the next section, we turn to the details of the top tree. In section 3.2, we specify exactly how we store the collections of group arrays in the middle and bottom levels, which depends on exactly which bounds we desire. In section 3.3, we put the pieces together to obtain the final algorithms and analysis.

3.1. Splits and merges. At a high level, our algorithms for searching, inserting, and deleting in the top tree follow the corresponding algorithms for a weight-balanced B-tree from section 2.2. The search algorithm is identical, and costs $O(\log_{B+1} N)$ memory transfers because of Lemma 2 bounding the cost in a van Emde Boas layout, and because replacing an array with a packed-memory array does not increase the number of memory transfers by more than a constant factor. The insertion and deletion algorithms must pay careful attention to maintain the van Emde Boas order, which sometimes changes drastically as the result of a split or merge.

An insertion or deletion in a weight-balanced B-tree consists of splits and merges along a leaf-to-root path, starting at a leaf and ending at a node at some height. We show how to split or merge a node at height h in the top tree using $O(1 + d^h/B)$ memory transfers (which we call the *split-merge cost*), plus the memory transfers incurred by a single packed-memory insertion or deletion (including updating pointers). By the amortization property of weight-balanced B-trees (condition 4 of Definition 3), it follows that the amortized split-merge cost of rebalancing a node v is $O(1/B)$ memory transfers per insertion or deletion into the subtree rooted at v . When we insert or delete an element, this element is added or removed in $O(\log N)$ such subtrees. Hence, the split-merge cost of an update is $O(1 + \frac{\log N}{B})$ amortized memory transfers.

Next we describe the algorithm to split a node v . First, we insert a new node v' into the packed-memory array immediately after v . Then we redistribute the children pointers among v and v' according to the split algorithm of weight-balanced B-trees, using $O(1)$ memory transfers. If v is the root of the tree, we also insert a new root node at the beginning of the packed-memory array and add parent-child pointers connecting this node to v and v' . Because of the rounding scheme in the van Emde Boas layout, this change in the height of the tree does not change the layout. However, adding v' and redistributing v 's children change the van Emde Boas layout significantly.

To see how to repair the van Emde Boas layout, consider the coarsest level of detail in which v is the root of a recursive subtree S . Suppose S has height h' , which can be only smaller than the height h of node v . Let S be composed of a top recursive subtree A of height $h' - \lfloor\lfloor h'/2 \rfloor\rfloor$ and bottom recursive subtrees B_1, B_2, \dots, B_k each of height $\lfloor\lfloor h'/2 \rfloor\rfloor$; refer to Figure 3. The split algorithm recursively splits A into A' and A'' . (In the base case, A is the singleton tree $\{v\}$, which we have already split into $\{v\}$ and $\{v'\}$.)

At this point, A' and A'' are next to each other. Now we must move them to the appropriate locations in the van Emde Boas layout. Let B_1, B_2, \dots, B_i be the children recursive subtrees of A' , and let $B_{i+1}, B_{i+2}, \dots, B_k$ be the children recursive subtrees of A'' . We need to move B_1, B_2, \dots, B_i in between A' and A'' . This move is accomplished by three linear scans. Specifically, we scan to copy A'' to some tem-

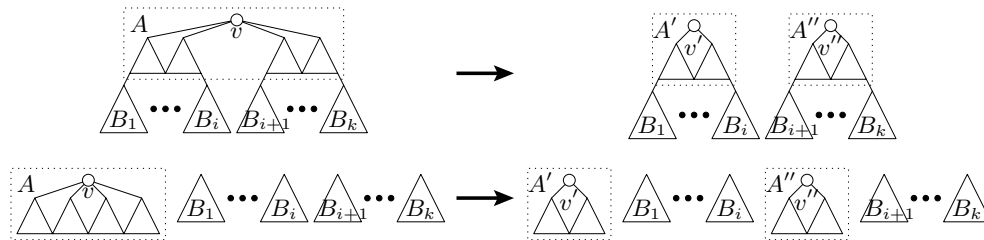


FIG. 3. *Splitting a node. The top shows the modification in the recursive subtree S , and the bottom shows the modification in the van Emde Boas layout.*

porary space, then scan to copy B_1, B_2, \dots, B_i immediately after A' , overwriting A'' , and then scan to copy the temporary space containing A'' to immediately after B_i .

Now that A' and B_1, B_2, \dots, B_i have been moved, we need to update the pointers to the nodes in these blocks. First, we scan through the nodes in A' and update the children pointers of the leaves to point to the new locations of B_1, B_2, \dots, B_i . That is, we increase the pointers by $\|A'\|$, the amount of space occupied in the packed-memory array by A' , including unused nodes. Second, we update the parent pointers of $B_{i+1}, B_{i+2}, \dots, B_k$ to A'' , decreasing them by $\|B_1\| + \|B_2\| + \dots + \|B_k\|$. Finally, we scan the recursive subtrees of height h' that are children of B_1, B_2, \dots, B_i and update the parent pointers of the roots, decreasing them by $\|A''\|$. This update can be done in a single scan because the children recursive subtrees of B_1, B_2, \dots, B_i are stored contiguously.

Finally, we analyze the number of memory transfers made by moving blocks at all levels of detail. At each level h' of the recursion, we perform a scan of all the nodes at most six times (three for the move, and three for the pointer updates). By strong weight balance (Property 2 and condition 3 of Definition 3), these scans cost at most $O(1 + d^{h'}/B)$ memory transfers. The total split-merge cost is given by the cost of recursing on the top recursive subtree of at most half the height and by the cost of the six scans. This recurrence is dominated by the top level:

$$T(h') \leq T\left(\frac{h'}{2}\right) + c\left(1 + \frac{d^{h'}}{B}\right) \leq c\left(1 + \frac{d^{h'}}{B}\right) + O\left(\frac{d^{h'/2}}{B}\right).$$

Hence, the split cost is $O(1 + d^{h'}/B) \leq O(1 + d^h/B)$ memory transfers (not counting the cost of a packed-memory insertion).

A merge can be performed within the same memory-transfer bound by using the same overall algorithm. To begin, we merge two nodes v and v' and apply a packed-memory deletion. In each step of the recursion, we perform the above algorithm in reverse, i.e., the opposite transformation from Figure 3. Therefore, the split-merge cost in either case is $O(1 + d^h/B)$ worst-case memory transfers per split or merge, which amortizes to $O(1 + \frac{\log N}{B})$ memory transfers per insertion or deletion.

The remaining cost per insertion and deletion is the cost to insert or delete an element from the packed-memory array, including the cost of updating pointers. By Theorem 5, the cost to rearrange the array itself is $O(1 + \frac{\log^2 N}{B})$ amortized memory transfers. Unfortunately, as mentioned above, the cost to update the pointers to the amortized $O(\log^2 N)$ elements moved can cost $O(\log^2 N)$ memory transfers. Therefore the total cost per insertion or deletion is $O(\log^2 N)$ amortized memory transfers, proving the following intermediate result.

LEMMA 6. *The top tree maintains an ordered set subject to searches in $O(1 +$*

$\log_{B+1} N$) memory transfers and to insertions and deletions in $O(\log^2 N)$ amortized memory transfers.

3.2. Using indirection. We use two levels of indirection to reduce the cost of updating pointers in the top tree of the previous section. The resulting data structure has three layers.

The bottom layer stores all N elements, clustered into $\Theta(N / \log N)$ groups of $\Theta(\log N)$ consecutive elements each. Associated with each of these bottom groups is a *representative element*, the minimum element in the group. A representative element may in fact be a *ghost element* which has been deleted but cannot be removed from the structure because it is used in higher layers.

The middle layer stores the $\Theta(N / \log N)$ representative elements from the bottom layer (some of which may be ghost elements). The middle layer may also store ghost elements which have been deleted from the bottom layer but are still representatives in the middle layer. The middle elements are clustered into $\Theta(N / \log^2 N)$ groups of $\Theta(\log N)$ consecutive elements each. Again, we elect the minimum element of each middle group as a representative element.

The top layer stores the $\Theta(N / \log^2 N)$ representative elements from the middle layer (some of which may be ghost elements).

The layers are stored according to the following data structures. The top layer is implemented by the top tree from the previous section. The middle layer is implemented by a single packed-memory structure, where the representative elements serve as markers between groups. The bottom layer is implemented by a packed-memory array if we require optimal scan operations. Otherwise, the bottom layer is implemented by an unordered collection of groups, where the elements in each group are stored in an arbitrary order within a contiguous region of memory. These memory regions are all of the same size and are allowed to be a constant fraction empty. Elements are inserted into or deleted from a group simply by rewriting the entire group.

Elements appearing on multiple layers have “down” pointers from each instantiation to the instantiation at the next lower layer. Elements appearing on both the top and middle layers have “up” pointers from the middle instantiation to the top instantiation. However, elements appearing on both the middle and bottom layers do not have up pointers from the bottom layer to the middle layer.

To search for an element in this data structure, we search in the top layer for the query element or, failing that, its predecessor. Then we follow the pointer to the corresponding group in the middle layer and scan through this middle group to find the query element or its predecessor. If the found element is a ghost element of the middle layer that has been deleted from the bottom layer, we move to the previous element in the middle layer. Finally, we follow the pointer to the corresponding group in the bottom layer and scan through this bottom group in search of the query element.

When an element is inserted, it is added to a bottom group according to order; if the inserted element could fit in two bottom groups, we favor the smaller of the two groups. Thus, a freshly inserted element never becomes the new representative element of its group. A group in the bottom or middle layer may become too full, in which case we split the group evenly into two groups, create a new representative element for the second group, and insert this representative element into the next level up. Similarly, a group in the bottom or middle layer may become too empty (by a fixed constant fraction less than $\frac{1}{2}$), in which case we merge the group with an adjacent group and delete the larger representative element from the next level up.

A merge may cause an immediate split.

3.3. Analysis. We detail the search, insert, and delete algorithms and analyze their performance in two versions of the data structure. The *ordered B-tree* stores the bottom layer as a packed-memory structure, and therefore supports scans optimally. The *unordered B-tree* stores the bottom layer as an unordered collection of groups.

LEMMA 7. *In both B-tree structures, a search uses $O(1 + \log_{B+1} N)$ memory transfers.*

Proof. A search examines each layer once from the top down. Searching through the tree in the top layer costs $O(1 + \log_{B+1} N)$ memory transfers by Lemma 6. Scanning through a group in the middle or bottom layer costs $O(1 + \frac{\log N}{B})$ memory transfers because each group contains $O(\log N)$ elements and is stored in a contiguous array of $O(\log N)$ elements. The cost at the top layer dominates. \square

LEMMA 8. *In the ordered B-tree, an update uses $O(1 + \log_{B+1} N + \frac{\log^2 N}{B})$ memory transfers.*

Proof. An update (insertion or deletion) examines each layer once from the bottom up. At the bottom layer, we pay $O(1 + \frac{\log^2 N}{B})$ amortized memory transfers according to Theorem 5 to rebalance the packed-memory array and preserve constant-size gaps. Of the $O(\log^2 N)$ amortized nodes that move during this rebalance on the bottom layer, $O(\log N)$ amortized nodes are also present on the middle layer, and we must update the down pointers from these nodes on the middle layer to the moved nodes on the bottom layer. To update these pointers, we scan the bottom and middle layers in parallel, with the middle-layer scan advancing at a relative speed of $\Theta(1 / \log N)$. These pointer updates cost $O(1 + \frac{\log^2 N}{B})$ amortized memory transfers.

A bottom group causes a split or merge after $\Omega(\log N)$ updates to that group since the last split or merge. When such a split or merge occurs, we pay $O(1 + \frac{\log^2 N}{B})$ amortized memory transfers to rebalance the packed-memory array on the middle layer. This split-merge cost can be charged to the $\Omega(\log N)$ updates that caused it, reducing the amortized cost by a factor of $\Omega(\log N)$. Thus, we pay only $O(1 + \frac{\log N}{B})$ amortized memory transfers per update. Of the $O(\log^2 N)$ amortized nodes that move during this rebalance on the middle layer, $O(\log N)$ amortized nodes are also present on the top layer, and we must update the down pointers from these nodes on the top layer to the moved nodes on the middle layer. To update these pointers, we scan the moved nodes on the middle layer, follow each node's up pointer if it is present, and update the down pointer of each node reached. These pointer updates cost at most $O(\log N + \frac{\log^2 N}{B})$ amortized memory transfers per split or merge, or $O(1 + \frac{\log N}{B})$ per update. (At this point we could also afford to update up pointers from the bottom layer to the middle layer at an amortized cost of $\Theta(1 + \frac{\log^3 N}{B})$ per split or merge, or $\Theta(1 + \frac{\log^2 N}{B})$ per update, but we do not need these pointers and, furthermore, the unordered B-tree cannot afford to maintain them.)

A middle group causes a split or merge after $\Omega(\log N)$ updates to that group, which correspond to $\Omega(\log^2 N)$ updates to the data structure. When such a split or merge occurs, we pay $O(\log^2 N)$ amortized memory transfers to update the top layer, which is only $O(1)$ amortized memory transfers per update to the data structure. Furthermore, the number of nodes moved in the top layer is also $O(\log^2 N)$ amortized, so $O(\log^2 N)$ amortized up pointers from the middle layer to the top layer need to be updated. Thus the pointer-update cost is $O(\log^2 N)$ amortized memory transfers per split or merge, or $O(1)$ per update to the data structure.

Summing all costs, an update costs $O(1 + \frac{\log^2 N}{B})$ amortized memory transfers plus the cost of searching. \square

LEMMA 9. *In the unordered B-tree, an update uses $O(1 + \log_{B+1} N)$ amortized memory transfers.*

Proof. The analysis is nearly identical. The only difference is that we no longer rebalance a packed-memory array on the bottom layer, but instead maintain an unordered collection of contiguous groups. As a result, we do not pay $O(1 + \frac{\log^2 N}{B})$ amortized memory transfers per update, neither for the packed-memory update nor for updating down pointers from the middle layer to the bottom layer. The additional cost of rewriting a bottom group during each update is $O(1 + \frac{\log N}{B})$ memory transfers. The cost of splitting and/or merging a bottom group is the same. Therefore, the total cost of an update is $O(1 + \frac{\log N}{B})$ amortized memory transfers plus the cost of searching. \square

Combining Lemmas 7–9, we obtain the following main results.

THEOREM 10. *The ordered B-tree maintains an ordered set subject to searches in $O(1 + \log_{B+1} N)$ memory transfers, insertions and deletions in $O(1 + \log_{B+1} N + \frac{\log^2 N}{B})$ amortized memory transfers, and scanning S consecutive elements in $O(1 + S/B)$ memory transfers.*

THEOREM 11. *The unordered B-tree maintains an ordered set subject to searches in $O(1 + \log_{B+1} N)$ memory transfers and to insertions and deletions in $O(1 + \log_{B+1} N)$ amortized memory transfers.*

4. Extensions and alternative approaches. A previous version [18] of this paper presents a different cache-oblivious B-tree based on *buffer nodes*. One distinguishing aspect of this approach is that it uses no indirection, storing the tree in one packed-memory array. This B-tree achieves an update bound of $O(1 + \log_{B+1} N + \frac{\log B}{\sqrt{B}} \log^2 N)$ amortized memory transfers. With one level of indirection, the B-tree’s update bound reduces to $O(1 + \log_{B+1} N + \frac{\log^2 N}{B})$ amortized memory transfers while supporting optimal scans, or $O(1 + \log_{B+1} N)$ amortized memory transfers without optimal scans. Thus, we ultimately obtain the same bounds as the simpler B-trees presented above, though with one fewer level of indirection.

The basic idea of buffer nodes is to enable storing data of different “fluidity” in a single packed array, ranging from rapidly changing data that is cheap to update to slowly changing data that is expensive to update. In the context of trees, leaves are frequently updated but their pointers are to relatively nearby nodes, so updating these pointers is usually cheap, whereas nodes in, e.g., the middle level, are updated infrequently but their pointers are to disparate regions of memory. The buffer-node solution is to add a large number of extra (dataless) nodes in between data of different fluidity to “protect” the low-fluidity data from the frequent updates of high-fluidity data. In the context of trees, we add $O(N/\log N)$ buffer nodes in between the top recursive subtree A and bottom recursive subtrees B_1, B_2, \dots, B_ℓ . These buffers prevent the rebalance intervals in the packed-memory array from touching the nodes in A for a long time, leading to an $O(\frac{\log B}{\sqrt{B}} \log^2 N)$ term in the update bound.

For any of the cache-oblivious B-tree structures, a natural question is whether the $O(\frac{\log^2 N}{B})$ term in the update bound can be removed while still supporting scans optimally. Effectively, in addition to a search-tree structure, we need a linked-list data structure for supporting fast scans. Recently, Bender et al. [15] developed a cache-oblivious linked list that supports insertions and deletions in $O(1)$ memory transfers and scans of S consecutive elements in $O(1 + S/B)$ amortized memory transfers. Using

this linked list, they obtain the following optimal bounds for cache-oblivious B-trees.

THEOREM 12 (Bender et al. [15, Corollary 1]). *There is a cache-oblivious data structure that maintains an ordered set subject to searches in $O(\log_{B+1} N)$ memory transfers, insertions and deletions in $O(\log_{B+1} N)$ amortized memory transfers, and scanning S consecutive elements in $O(1 + S/B)$ amortized memory transfers.*

It remains open whether these bounds can be achieved in the worst case. See [15, 17] for cache-oblivious B-trees achieving some worst-case bounds.

5. Conclusion. We have presented cache-oblivious B-tree data structures that perform searches optimally. The first data structure maintains the data in sorted order in an array with gaps, and it stores an auxiliary structure for searching within this array. The second data structure attains the $O(\log_{B+1} N)$ update bounds of B-trees but breaks the sorted order of elements, thus slowing sequential scans of consecutive elements. These cache-oblivious B-trees represent the first dynamic cache-oblivious data structures. The cache-oblivious tools presented in this paper play an important role in the dynamic cache-oblivious data structures that have followed.

Acknowledgments. We gratefully acknowledge Charles Leiserson for suggesting this problem to us. We thank the anonymous referees for their many helpful comments.

REFERENCES

- [1] P. K. AGGARWAL, L. ARGE, A. DANNER, AND B. HOLLAND-MINKLEY, *Cache-oblivious data structures for orthogonal range searching*, in Proceedings of the 19th Annual ACM Symposium on Computational Geometry, San Diego, CA, 2003, pp. 237–245.
- [2] A. AGGARWAL, B. ALPERN, A. K. CHANDRA, AND M. SNIR, *A model for hierarchical memory*, in Proceedings of the 19th Annual ACM Symposium on Theory of Computing, New York, 1987, pp. 305–314.
- [3] A. AGGARWAL, A. K. CHANDRA, AND M. SNIR, *Hierarchical memory with block transfer*, in Proceedings of the 28th Annual IEEE Symposium on Foundations of Computer Science, Los Angeles, CA, 1987, pp. 204–216.
- [4] A. AGGARWAL AND J. S. VITTER, *The input/output complexity of sorting and related problems*, Comm. ACM, 31 (1988), pp. 1116–1127.
- [5] B. ALPERN, L. CARTER, E. FEIG, AND T. SELKER, *The uniform memory hierarchy model of computation*, Algorithmica, 12 (1994), pp. 72–109.
- [6] S. ALSTRUP, M. A. BENDER, E. D. DEMAINE, M. FARACH-COLTON, T. RAUHE, AND M. THORUP, *Efficient Tree Layout in a Multilevel Memory Hierarchy*, <http://www.arXiv.org/abs/cs.DS/0211010> (2004).
- [7] M. ANDREWS, M. A. BENDER, AND L. ZHANG, *New algorithms for the disk scheduling problem*, Algorithmica, 32 (2002), pp. 277–301.
- [8] L. ARGE, M. A. BENDER, E. D. DEMAINE, B. HOLLAND-MINKLEY, AND J. I. MUNRO, *Cache-oblivious priority queue and graph algorithm applications*, in Proceedings of the 34th Annual ACM Symposium on Theory of Computing, Montreal, Canada, 2002, pp. 268–276.
- [9] L. ARGE, G. S. BRODAL, R. FAGERBERG, AND M. LAUSTSEN, *Cache-oblivious planar orthogonal range searching and counting*, in Proceedings of the 21st Annual ACM Symposium on Computational Geometry, Pisa, Italy, 2005, pp. 160–169.
- [10] L. ARGE, M. DE BERG, AND H. HAVERKORT, *Cache-oblivious R-trees*, in Proceedings of the 21st Annual ACM Symposium on Computational Geometry, Pisa, Italy, 2005, pp. 170–179.
- [11] L. ARGE AND J. S. VITTER, *Optimal external memory interval management*, SIAM J. Comput., 32 (2003), pp. 1488–1508.
- [12] R. D. BARVE AND J. S. VITTER, *A theoretical framework for memory-adaptive algorithms*, in Proceedings of the 40th Annual IEEE Symposium on Foundations of Computer Science, New York, 1999, pp. 273–284.
- [13] R. BAYER AND E. M. MCCREIGHT, *Organization and maintenance of large ordered indexes*, Acta Inform., 1 (1972), pp. 173–189.

- [14] M. A. BENDER, G. S. BRODAL, R. FAGERBERG, D. GE, S. HE, H. HU, J. IACONO, AND A. LÓPEZ-ORTIZ, *The cost of cache-oblivious searching*, in Proceedings of the 44th Annual IEEE Symposium on Foundations of Computer Science, Cambridge, MA, 2003, pp. 271–282.
- [15] M. A. BENDER, R. COLE, E. D. DEMAINE, AND M. FARACH-COLTON, *Scanning and traversing: Maintaining data for traversals in a memory hierarchy*, in Proceedings of the 10th Annual European Symposium on Algorithms, Lecture Notes in Comput. Sci. 2461, Springer-Verlag, Berlin, 2002, pp. 139–151.
- [16] M. A. BENDER, R. COLE, E. D. DEMAINE, M. FARACH-COLTON, AND J. ZITO, *Two simplified algorithms for maintaining order in a list*, in Proceedings of the 10th Annual European Symposium on Algorithms, Lecture Notes in Comput. Sci. 2461, Springer-Verlag, Berlin, 2002, pp. 152–164.
- [17] M. A. BENDER, R. COLE, AND R. RAMAN, *Exponential structures for efficient cache-oblivious algorithms*, in Proceedings of the 29th International Colloquium on Automata, Languages and Programming, Lecture Notes in Comput. Sci. 2380, Springer-Verlag, Berlin, 2002, pp. 195–207.
- [18] M. A. BENDER, E. D. DEMAINE, AND M. FARACH-COLTON, *Cache-oblivious B-trees*, in Proceedings of the 41st Annual IEEE Symposium on Foundations of Computer Science, Redondo Beach, CA, 2000, pp. 399–409.
- [19] M. A. BENDER, E. D. DEMAINE, AND M. FARACH-COLTON, *Efficient tree layout in a multilevel memory hierarchy*, in Proceedings of the 10th Annual European Symposium on Algorithms, Lecture Notes in Comput. Sci. 2461, Springer-Verlag, Berlin, 2002, pp. 165–173.
- [20] M. A. BENDER, Z. DUAN, J. IACONO, AND J. WU, *A locality-preserving cache-oblivious dynamic dictionary*, *J. Algorithms*, 53 (2004), pp. 115–136.
- [21] R. D. BLUMOFE, M. FRIGO, C. F. JOERG, C. E. LEISERSON, AND K. H. RANDALL, *An analysis of dag-consistent distributed shared-memory algorithms*, in Proceedings of the 8th Annual ACM Symposium on Parallel Algorithms and Architectures, Padua, Italy, 1996, pp. 297–308.
- [22] G. S. BRODAL AND R. FAGERBERG, *Cache oblivious distribution sweeping*, in Proceedings of the 29th International Colloquium on Automata, Languages, and Programming, Lecture Notes in Comput. Sci. 2380, Springer-Verlag, Berlin, 2002, pp. 426–438.
- [23] G. S. BRODAL AND R. FAGERBERG, *Funnel heap—a cache oblivious priority queue*, in Proceedings of the 13th Annual International Symposium on Algorithms and Computation, Lecture Notes in Comput. Sci. 2518, Springer-Verlag, Berlin, 2002, pp. 219–228.
- [24] G. S. BRODAL AND R. FAGERBERG, *On the limits of cache-obliviousness*, in Proceedings of the 35th Annual ACM Symposium on Theory of Computing, San Diego, CA, 2003, pp. 307–315.
- [25] G. S. BRODAL, R. FAGERBERG, AND R. JACOB, *Cache oblivious search trees via binary trees of small height*, in Proceedings of the 13th Annual ACM-SIAM Symposium on Discrete Algorithms (San Francisco, CA), SIAM, Philadelphia, 2002, pp. 39–48.
- [26] G. S. BRODAL, R. FAGERBERG, U. MEYER, AND N. ZEH, *Cache-oblivious data structures and algorithms for undirected breadth-first search and shortest paths*, in Proceedings of the 9th Scandinavian Workshop on Algorithm Theory, Lecture Notes in Comput. Sci. 3111, Springer, 2004, pp. 480–492.
- [27] COMPAQ, *Documentation Library*, <http://ftp.digital.com/pub/Digital/info/semiconductor/literature/dsc-library.html> (1999).
- [28] P. F. DIETZ AND D. D. SLEATOR, *Two algorithms for maintaining order in a list*, in Proceedings of the 19th Annual ACM Symposium on Theory of Computing, New York, 1987, pp. 365–372.
- [29] G. FRANCESCHINI AND R. GROSSI, *Optimal cache-oblivious implicit dictionaries*, in Proceedings of the 30th International Colloquium on Automata, Languages, and Programming, Lecture Notes in Comput. Sci. 2719, Springer-Verlag, Berlin, 2003, pp. 316–331.
- [30] G. FRANCESCHINI AND R. GROSSI, *Optimal worst-case operations for implicit cache-oblivious search trees*, in Proceedings of the 8th Workshop on Algorithms and Data Structures, Lecture Notes in Comput. Sci. 2748, Springer-Verlag, Berlin, pp. 114–126.
- [31] M. FRIGO, C. E. LEISERSON, H. PROKOP, AND S. RAMACHANDRAN, *Cache-oblivious algorithms*, in Proceedings of the 40th Annual IEEE Symposium on Foundations of Computer Science, New York, 1999, pp. 285–297.
- [32] J.-W. HONG AND H. T. KUNG, *I/O complexity: The red-blue pebble game*, in Proceedings of the 13th Annual ACM Symposium on Theory of Computing, Milwaukee, WI, 1981, pp. 326–333.
- [33] A. ITAI, A. G. KONHEIM, AND M. RODEH, *A sparse table implementation of priority queues*, in Proceedings of the 8th Colloquium on Automata, Languages, and Programming, S. Even and O. Kariv, eds., Lecture Notes in Comput. Sci. 115, Springer-Verlag, Berlin, 1981,

- pp. 417–431.
- [34] P. KUMAR AND E. RAMOS, *I/O-Efficient Construction of Voronoi Diagrams*, manuscript, 2003.
 - [35] R. E. LADNER, R. FORTNA, AND B.-H. NGUYEN, *A comparison of cache aware and cache oblivious static search trees using program instrumentation*, in *Experimental Algorithmics: From Algorithm Design to Robust and Efficient Software*, Lecture Notes in Comput. Sci. 2547, Springer-Verlag, Berlin, 2002, pp. 78–92.
 - [36] A. LAMARCA AND R. E. LADNER, *The influence of caches on the performance of sorting*, *J. Algorithms*, 31 (1999), pp. 66–104.
 - [37] J. I. MUNRO, T. PAPADAKIS, AND R. SEDGEWICK, *Deterministic skip lists*, in *Proceedings of the 3rd Annual ACM-SIAM Symposium on Discrete Algorithms (Orlando, FL)*, SIAM, Philadelphia, 1992, pp. 367–375.
 - [38] J. NIEVERGELT AND E. M. REINGOLD, *Binary search trees of bounded balance*, *SIAM J. Comput.*, 2 (1973), pp. 33–43.
 - [39] M. H. OVERMARS, *The Design of Dynamic Data Structures*, Lecture Notes in Comput. Sci. 156, Springer-Verlag, Berlin, 1983.
 - [40] H. PROKOP, *Cache-Oblivious Algorithms*, Master’s thesis, Massachusetts Institute of Technology, Cambridge, MA, 1999.
 - [41] W. PUGH, *Skip lists: A probabilistic alternative to balanced trees*, in *Proceedings of the Workshop on Algorithms and Data Structures*, Lecture Notes in Comput. Sci. 382, Springer-Verlag, Berlin, 1989, pp. 437–449.
 - [42] N. RAHMAN, R. COLE, AND R. RAMAN, *Optimised predecessor data structures for internal memory*, in *Proceedings of the 5th International Workshop on Algorithm Engineering*, Lecture Notes in Comput. Sci. 2141, Springer-Verlag, Berlin, 2001, pp. 67–78.
 - [43] C. RUEMLER AND J. WILKES, *An introduction to disk drive modeling*, *IEEE Comput.*, 27 (1994), pp. 17–29.
 - [44] J. E. SAVAGE, *Extending the Hong-Kung model to memory hierarchies*, in *Proceedings of the 1st Annual International Conference on Computing and Combinatorics*, Lecture Notes in Comput. Sci. 959, Springer-Verlag, Berlin, 1995, pp. 270–281.
 - [45] S. SEN AND S. CHATTERJEE, *Towards a theory of cache-efficient algorithms*, in *Proceedings of the 11th Annual ACM-SIAM Symposium on Discrete Algorithms (San Francisco, CA)*, SIAM, Philadelphia, 2000, pp. 829–838.
 - [46] S. TOLEDO, *Locality of reference in LU decomposition with partial pivoting*, *SIAM J. Matrix Anal. Appl.*, 18 (1997), pp. 1065–1081.
 - [47] P. VAN EMDE BOAS, *Preserving order in a forest in less than logarithmic time*, in *Proceedings of the 16th Annual IEEE Symposium on Foundations of Computer Science*, Berkeley, CA, 1975, pp. 75–84.
 - [48] P. VAN EMDE BOAS, R. KAAS, AND E. ZIJLSTRA, *Design and implementation of an efficient priority queue*, *Math. Systems Theory*, 10 (1977), pp. 99–127.
 - [49] J. S. VITTER, *External memory algorithms and data structures: Dealing with massive data*, *ACM Comput. Surveys*, 33 (2001), pp. 209–271.
 - [50] J. S. VITTER AND E. A. M. SHRIVER, *Algorithms for parallel memory I: Two-level memories*, *Algorithmica*, 12 (1994), pp. 110–147.
 - [51] J. S. VITTER AND E. A. M. SHRIVER, *Algorithms for parallel memory II: Hierarchical multilevel memories*, *Algorithmica*, 12 (1994), pp. 148–169.
 - [52] D. E. WILLARD, *Maintaining dense sequential files in a dynamic environment*, in *Proceedings of the 14th Annual ACM Symposium on Theory of Computing*, San Francisco, CA, 1982, pp. 114–121.
 - [53] D. E. WILLARD, *Good worst-case algorithms for inserting and deleting records in dense sequential files*, in *Proceedings of the 1986 ACM SIGMOD International Conference on Management of Data*, Washington, DC, 1986, pp. 251–260.
 - [54] D. E. WILLARD, *A density control algorithm for doing insertions and deletions in a sequentially ordered file in good worst-case time*, *Inform. and Comput.*, 97 (1992), pp. 150–204.

A POWERFUL $LL(k)$ COVERING TRANSFORMATION*

GYUNG-OK LEE[†] AND KWANG-MOO CHOE[‡]

Abstract. k -transformable grammars have been conjectured to be the uppermost class of $LL(k)$ covering transformable grammars. $PLR(k)$ grammars have been known as a well characterized subclass of k -transformable grammars. Being contrary to those claims, this paper shows that some $PLR(k)$ grammars are not k -transformable, and so k -transformable grammars are not the true uppermost.

A powerful $LL(k)$ covering transformation is suggested in this paper. It is a generalization of the transformations of k -transformable grammars and $PLR(k)$ grammars. A remarkable aspect of the new transforming process is the deterministic property, where “deterministic” means that the transformation is obtained in a single process without requiring any heuristic, unlike k -transformable grammars’ transformation for which a heuristic is required. The transformable grammar class is shown to be larger than k -transformable grammars and $PLR(k)$ grammars.

Key words. compilers, LR grammars, LL grammars, left-to-right cover

AMS subject classifications. 68N20, 68Q42, 68Q45

DOI. 10.1137/S0097539701400154

1. Introduction. LR grammars and LL grammars are two of the most representative context-free grammars for programming languages. When we compare the parsers of those grammars, we see that LL parsers are easier and simpler to implement, understand, and error recover and have a greater flexibility in computing semantic attributes during parsing, but LL grammars are a small subclass of LR grammars. Some researchers thus suggested generalized LL parsing techniques [5, 8, 11], which are more applicable to the larger grammar class than the class of LL grammars. On the other hand, the restricted $LR(k)$ grammars, k -transformable grammars [3, 4], and $PLR(k)$ grammars [10] were suggested by Hammer and Soisalon-Soininen, respectively, as the grammars able to be transformed into $LL(k)$ covering grammars. Specially, k -transformable grammars have been conjectured as the uppermost class of such $LR(k)$ grammars [3], and $PLR(k)$ grammars have been known as a well-characterized subclass of k -transformable grammars [10].

In this paper we are interested in the covering transformation of LR grammars into LL grammars. The main idea in the transformation starts from the prediction of reduction goals during LR parsing. In LR parsing, a nonterminal, to which some prefix of the remaining input is reduced, is known at the reduction time, but we can often find a reduction goal before that time. A nonterminal B and a suffix γ of α are *predicted*, when the stack string is α , if it is certain that γ and some prefix of the remaining input will be reduced to B . (That is, the left side of Figure 1.1 is expected to be the right side.) In Hammer’s method, γ is restricted to be ϵ , and in Soisalon-Soininen’s method, γ has to be the left corner symbol of the right side of a production. Soisalon-Soininen’s γ gives more information about B than Hammer’s γ and, as a result, some predictable goals by Soisalon-Soininen’s method cannot be

*Received by the editors December 23, 2001; accepted for publication (in revised form) May 2, 2005; published electronically October 17, 2005.

<http://www.siam.org/journals/sicomp/35-2/40015.html>

[†]Department of Information Science and Telecommunications, Hanshin University, 411 Yangsan-dong, Osan, Kyunggi-Do 447-791, Republic of Korea (golee@hanshin.ac.kr).

[‡]Department of Computer Science, Korea Advanced Institute of Science and Technology, 373-1 Kusung-dong, Yuseong-ku, Taejeon 305-701, Republic of Korea (choe@cs.kaist.ac.kr).

predicted when Hammer's method is applied. We found that the lack of predictability causes the class of k -transformable grammars not to completely include the class of $PLR(k)$ grammars.

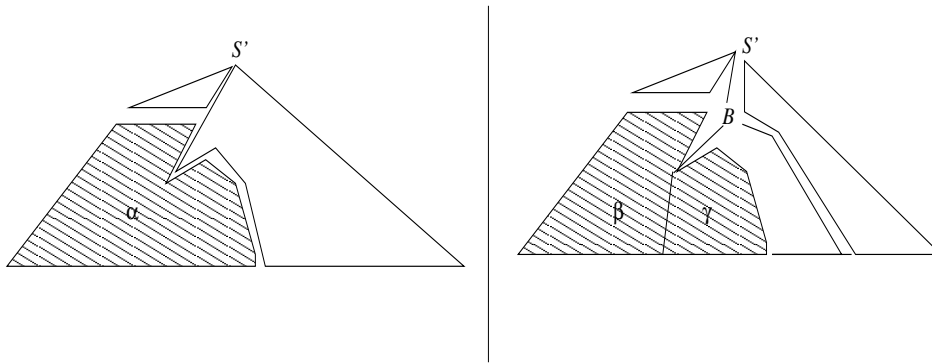


FIG. 1.1. Prediction in LR parsing ($\alpha = \beta\gamma$).

This paper generalizes the previous prediction technique into two viewpoints. The first key idea is to predict a reduction goal after enough information for the goal is known. The delay in predicting time obviously enlarges the range of predictable goals. In this paper, we allow γ in the right side of Figure 1.1 to be an arbitrary string. On the other hand, in both Hammer and Soisalon-Soininen's methods, the prediction of goal B at a parsing time implies that whenever a generatable string from B appears as the k -length prefix of the remaining input, a reduction to B will certainly occur. The second key idea is to allow the prediction to be performed not only over the total string but also over some partial string generatable from a predicted goal. That is, B can be a predicted goal as long as any predictive string exists, although some generatable strings from B do not guarantee a reduction to B .

This paper suggests a powerful covering transformation into $LL(k)$ form based on the generalized prediction. The applicable grammar class is defined as *extended $PLR(k)$ grammars*. They are larger than k -transformable grammars and $PLR(k)$ grammars. Hammer informally described an extension [3, pp. 289–293] of k -transformable grammars. After his argument, there has been no more research on the extension. The main reason is considered to be intricateness of the construction of multiple stack machine [3], which is required to obtain an $LL(k)$ transformation. (Actually, Hammer was worried that the extension needs numerous complications in his original machine model [3, p. 292, lines 1–3].) On the other hand, recently we suggested a grammatical characterization of k -transformable grammars in [7]. This paper develops an extension of k -transformable grammars using grammatical derivation. We believe that our extension completely includes the informal extension of Hammer.

Another contribution of this paper is the deterministic selection of the new transformation compared with the nondeterministic one of Hammer's method, where "deterministic" means that an LL covering grammar can be obtained in a single process, whereas in Hammer's method a transformer has to choose a set of predictable goals using a heuristic. As a result, whether a grammar is transformable or not can be decided in a single process. The grammatical characterization of the transformable

grammars is also given.

Section 2 contains basic notation and definitions that are used in the subsequent sections. The counterexample showing that a PLR(k) grammar is not k -transformable is presented in section 3. Section 4 defines two relations, which are used to describe the generalized prediction. The idea for the deterministic transforming is developed in section 5. Section 6 suggests a new LL(k) covering transformation and relates the transformable grammars with k -transformable grammars and PLR(k) grammars. Finally, section 7 summarizes this paper.

2. Basic notation and definitions. We use notation and definitions based on [9], and the reader is assumed to be familiar with them.

Throughout this paper, the symbol G denotes an arbitrary context-free grammar $G = (N, \Sigma, P, S)$ (let $V = N \cup \Sigma$). The symbol k represents an arbitrary positive integer. Without being explicitly stated, G is assumed to be LR(k). Let $S' \notin N$ and a terminal symbol $\$ \notin \Sigma$. Then $G' = (N \cup \{S'\}, \Sigma \cup \{\$\}, P \cup \{S' \rightarrow S\$^k\}, S')$ is the augmented grammar of G . In this paper, G is assumed to be augmented and *reduced* [9].

Lowercase Greek letters, such as $\alpha, \beta,$ and $\gamma,$ denote strings in V^* ; lowercase Roman letters near the beginning of the alphabet, such as $a, b,$ and $c,$ are in $\Sigma,$ and those near the end, such as $w, x, y,$ and $z,$ are in $\Sigma^*;$ uppercase Roman letters near the end of the alphabet, such as $W, X,$ and $Y,$ are in $V.$ The empty string is denoted by $\epsilon.$ The reverse of a string α is represented by $\alpha^R.$

Let w be a terminal string. Then $k:w$ is equal to w if $|w| \leq k,$ or the first k symbols of w otherwise. For any G and any $\alpha \in (V \cup \{\$\})^*,$ $L^G(\alpha) = \{x | \alpha \Rightarrow^* x \text{ in } G, x \in \Sigma^*\},$ $FIRST_k^G(\alpha) = \{k:x | \alpha \Rightarrow^* x \text{ in } G, x \in \Sigma^*\},$ and $FOLLOW_k^G(\alpha) = \{k:x | S' \Rightarrow^* \beta\alpha x \text{ in } G, x \in \Sigma^*\}.$ If G is obvious, then it is omitted.

A string $\alpha \in V^*$ is said to be a *viable prefix of G* if there exists a derivation $S' \Rightarrow_{rm}^* \beta Bz \Rightarrow_{rm} \beta\gamma\delta z \Rightarrow_{rm}^* \beta\gamma yz$ in $G,$ where $\beta\gamma = \alpha$ and rm stands for the rightmost derivation. For a viable prefix α of $G,$ *k -right context* is defined as: $RC_k^G(\alpha) = \{k:yz | \text{there exists } S' \Rightarrow_{rm}^* \beta Bz \Rightarrow_{rm} \beta\gamma\delta z \Rightarrow_{rm}^* \beta\gamma yz \text{ in } G \text{ where } \beta\gamma = \alpha\}.$

A restricted relation $\Rightarrow_{A,R}$ of $\Rightarrow_{rm},$ where $A \in N$ and $R \subseteq FOLLOW_k(A),$ is defined. Let $p = B \rightarrow X\delta.$ Suppose that there exists $Ar \Rightarrow_{rm}^* \gamma Bzr \Rightarrow_{rm}^p \gamma X\delta zr,$ where $r \in R$ and $z \in \Sigma^*.$ Then

$$\gamma Bzr \Rightarrow_{A,R}^p \gamma X\delta zr \begin{cases} \text{holds when } FIRST_k(\delta zr) \cap R = \emptyset & \text{if } \gamma = \epsilon \text{ and } X = A, \\ \text{always holds} & \text{otherwise.} \end{cases}$$

Every derivation using $\Rightarrow_{A,R}$ starts from A and must not derive a string containing A at the leftmost position such that the string of the A immediately following belongs to $R.$ That is, the derivation $Ar \Rightarrow_{A,R}^+ Axr,$ where $r \in R$ and $k:xr \in R$ is not allowed. The difference between $\Rightarrow_{A,R}$ and \Rightarrow_{rm} is clear when A is left recursive.

The k -right context function RC_k^G is localized over derivation $\Rightarrow_{A,R}$ as follows. Let $\alpha \in V^*.$ Then $RC_k^{A,R}(\alpha) = \{k:yzr | \text{there exists } Ar \Rightarrow_{A,R}^* \beta Bzr \Rightarrow_{A,R} \beta\gamma\delta zr \Rightarrow_{A,R}^* \beta\gamma yzr \text{ in } G \text{ where } r \in R \text{ and } \beta\gamma = \alpha\}.$

The equivalence of grammars can be considered in terms of languages or syntactic structures. That is, for G_1 and $G_2,$ the former means $L(G_1) = L(G_2),$ and the latter means that the grammars satisfy a covering property. We present the definition of left-to-right cover.

DEFINITION 2.1. Let $G_1 = (N_1, \Sigma, P_1, S_1)$ and $G_2 = (N_2, \Sigma, P_2, S_2)$ be grammars such that $L(G_1) = L(G_2),$ and let h be a homomorphism from P_2 to $P_1.$

Then we say that G_2 left-to-right covers G_1 with respect to h if the following conditions are satisfied:

1. If there exists $S_2 \Rightarrow_{l_m}^\pi w$ in G_2 , then there exists $S_1 \Rightarrow_{r_m}^{h(\pi)} w$ in G_1 , and
2. if there exists $S_1 \Rightarrow_{r_m}^\pi w$ in G_1 , then there exists $S_2 \Rightarrow_{l_m}^\pi w$ in G_2 , where $h(\pi) = \pi'$.

Let G_2 be $LL(k)$ and G_1 be $LR(k)$. Then G_2 *LL-to-LR(k)* covers G_1 if G_2 left-to-right covers G_1 with respect to h for some h . At this time, G_2 is said to be an *LL-to-LR(k) covering* (or simply, *LL(k) covering*) grammar of G_1 .

We next present the definition of $PLR(k)$ grammars.

DEFINITION 2.2. (see [10]). A grammar $G = (N, \Sigma, P, S)$ is said to be *PLR(k)* if in G' , for each production $A \rightarrow X\delta$, where $X\delta \neq \epsilon$, the conditions

$$\begin{aligned} S' &\Rightarrow_{r_m}^* \beta A z_1 \Rightarrow_{r_m} \beta X \delta z_1 \Rightarrow_{r_m}^* \beta X y_1 z_1, \\ S' &\Rightarrow_{r_m}^* \beta' B z_2 \Rightarrow_{r_m} \beta' \beta'' X \zeta z_2 \Rightarrow_{r_m}^* \beta' \beta'' X y_2 z_2, \text{ and} \\ &\beta' \beta'' = \beta \text{ and } k:y_1 z_1 = k:y_2 z_2 \end{aligned}$$

always imply that $\beta A = \beta' B$.

Instead of the original definition [3] of k -transformable grammars, we use the following theorem to characterize k -transformable grammars because it does not require any understanding of the intricate multiple stack machine.

THEOREM 2.3. (see [7]). G is k -transformable iff there exists a constant n , depending on G , such that if $\alpha (\neq \epsilon)$ is a viable prefix of G and v is a string in $RC_k^G(\alpha)$, then there exist B, W , and $\gamma (\neq \epsilon)$, where $\alpha = \beta\gamma$, $|\gamma| \leq n$, and $v \in RC_k^{B,W}(\gamma)$ such that whenever there exists $S' \Rightarrow_{r_m}^* \beta\gamma z \Rightarrow_{r_m}^* \beta y z$ in G , where $k:yz \in RC_k^{B,W}(\epsilon)$, there exist $S' \Rightarrow_{r_m}^* \beta B z''$ and $Bw \Rightarrow_{B,W}^* \gamma z'w \Rightarrow_{B,W}^* yz'w$ in G , where $w = k:z'' (w \in W)$ and $z'z'' = z$. (We call the n the characteristic of k -transformableness for G .)

3. A counterexample. The following example shows a $PLR(1)$ grammar that is not k -transformable for all $k \geq 0$.

Example 3.1. Let $G1 = (\{S, C, D, X, Y\}, \{c, d, b, a\}, \{S \rightarrow C, C \rightarrow DXc, C \rightarrow DYd, D \rightarrow bD, D \rightarrow a, X \rightarrow DC, X \rightarrow DD, Y \rightarrow Dd\}, S)$. Then we can decide that $G1$ is $LR(1)$ and $PLR(1)$, but is not k -transformable for all $k (k \geq 1)$. Suppose that $G1$ is k -transformable for some fixed k and n is the characteristic of k -transformableness for $G1$. Let us set α and v to be $DD(DD)^j$ for some $j > n$ and b , respectively. Then we cannot find the B, W , and γ in Theorem 2.3. Hence, $G1$ cannot be k -transformable.

Thus, $PLR(k)$ grammars are not a subclass of k -transformable grammars. Conversely, k -transformable grammars are not a subclass of $PLR(k)$ grammars. For example, a grammar with the production set $\{S \rightarrow aAa, A \rightarrow Bb, B \rightarrow Aa, A \rightarrow b\}$ is $LR(1)$ and 1-transformable. The grammar, however, is not $PLR(1)$ because there exist two contradictory derivations $S' \Rightarrow_{r_m} S\$ \Rightarrow_{r_m} aAa\$$ and $S' \Rightarrow_{r_m} S\$ \Rightarrow_{r_m} aAa\$ \Rightarrow_{r_m} aBba\$ \Rightarrow_{r_m} aAaba\$$. Thus, there is no inclusion relationship between k -transformable grammars and $PLR(k)$ grammars. A generalized grammar class, which contains both grammars, is hence motivated, and the subsequent sections treat this subject.

4. Two relations. This section defines two relations in order to formally describe the generalized prediction.

4.1. d relation. A rightmost derivation in a grammar can be analyzed through the dependency relation of Knuth [6]. The relation, called d relation, is here refined

by attributing some terminal strings in order to represent context information in sentential forms.

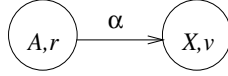
Let $A \in N$, $r \in \Sigma^k$, $\alpha \in V^*$, $X \in V \cup \{\epsilon\}$, and $v \in \Sigma^k \cup \Sigma^{k-1}$. Then

$$(4.1) \quad (A, r) d^\alpha (X, v) \text{ iff } A \rightarrow \alpha X \beta \in P, r \in FOLLOW_k(A),$$

$v \in FIRST_k(\beta r)$ when $X \in N \cup \{\epsilon\}$, and $v \in FIRST_{k-1}(\beta r)$ when $X \in \Sigma$.

Let $(A, r) d^{0^\epsilon} (A, r)$ be $(A, r) d^\epsilon (A, r)$, and $(A, r) d^{n^\alpha} (X, v)$, $n > 0$, be the composition of $(A, r) d^{n-1^\beta} (B, w)$ and $(B, w) d^\gamma (X, v)$, where $\beta\gamma = \alpha$. The reflexive transitive closure of d relation, denoted by d^* , is defined by $\bigcup_{n \geq 0} d^n$.

The directed graph associated with the d relation is called the d -graph. It is constructed by representing each instance (4.1) as a pair of vertices connected by a directed edge:



A path is represented by $h = (A_0, r_0) d^{\alpha_1} (A_1, r_1) \cdots (A_{n-1}, r_{n-1}) d^{\alpha_n} (A_n, r_n)$. If $A_n \in \Sigma$ or $A_n = \epsilon$, then h is a *terminal path*. If the intercomponents $(A_1, r_1), \dots, (A_{n-1}, r_{n-1})$ are not important, then h is represented by $(A_0, r_0) d^{*\alpha_1 \cdots \alpha_n} (A_n, r_n)$.

Given a path in the d -graph, we can infer a corresponding rightmost derivation, and vice versa.

PROPERTY 4.1. *Let $A_i \in N$ and $r_i \in \Sigma^k$, $i = 0, 1, \dots, n$. Then there exists a path $h = (A_0, r_0) d^{\alpha_1} (A_1, r_1) d^{\alpha_2} (A_2, r_2) \cdots (A_{n-1}, r_{n-1}) d^{\alpha_n} (A_n, r_n)$ in the d -graph iff there exists a derivation in G such that $A_0 r_0 \Rightarrow_{rm}^* \alpha_1 A_1 \beta_1 r_0 \Rightarrow_{rm}^* \alpha_1 A_1 z_1 r_0 \Rightarrow_{rm}^* \alpha_1 \alpha_2 A_2 \beta_2 z_1 r_0 \Rightarrow_{rm}^* \alpha_1 \alpha_2 \cdots \alpha_{n-1} A_{n-1} z_{n-1} \cdots z_1 r_0 \Rightarrow_{rm}^* \alpha_1 \alpha_2 \cdots \alpha_{n-1} \alpha_n A_n \beta_n z_{n-1} \cdots z_1 r_0 \Rightarrow_{rm}^* \alpha_1 \alpha_2 \cdots \alpha_{n-1} \alpha_n A_n z_n z_{n-1} \cdots z_1 r_0$ for some $z_i \in \Sigma^*$ where $k: z_i \cdots z_1 r_0 = r_i$ ($i = 1, \dots, n$).*

Proof. The *only if* part can be proved by simple induction on n , and the *if* part can be proved by directly applying the definition of the d relation. \square

A sequence of d^ϵ -related vertices in a path is defined as a *segment*. Let $h = (A_0, r_0) d^{\alpha_1} (A_1, r_1) \cdots (A_{n-1}, r_{n-1}) d^{\alpha_n} (A_n, r_n)$ be a path in the d -graph. If $\beta = \alpha_1 \cdots \alpha_i$ and $\alpha_{i+1} \cdots \alpha_j = \epsilon$, then the $|\beta|$ -segment of h is $(A_i, r_i) d^{\alpha_{i+1}} (A_{i+1}, r_{i+1}) \cdots (A_{j-1}, r_{j-1}) d^{\alpha_j} (A_j, r_j)$, where

$$(4.2) \quad \alpha_i \neq \epsilon \text{ when } i \neq 0 \text{ and } \alpha_{j+1} \neq \epsilon \text{ when } j \neq n.$$

Condition (4.2) says that the immediate predecessor and the immediate successor of a segment, if they exist, are not d^ϵ -related.

PROPERTY 4.2. *Let $A \in N$ and $r \in \Sigma^k$. There exists $(A_l, r_l) = (A, r)$ for some l ($i \leq l \leq j$) in the $|\beta|$ -segment $(A_i, r_i) d^\epsilon (A_{i+1}, r_{i+1}) \cdots (A_{j-1}, r_{j-1}) d^\epsilon (A_j, r_j)$ of the path $(A_0, r_0) d^{*\alpha} (A_n, r_n)$ iff there exists $A_0 r_0 \Rightarrow_{rm}^* \beta A z r_0$ in G , where $k: z r_0 = r$.*

We next define a specialized path in the d -graph that is related to a derivation over $\Rightarrow_{A, R}$.

DEFINITION 4.1. *Let $A \in N, r \in \Sigma^k, \alpha \in V^*$, and $u \in \Sigma^k$. Suppose that $h = (A_0, r_0) d^{\alpha_1} (A_1, r_1) \cdots (A_{n-1}, r_{n-1}) d^{\alpha_n} (A_n, r_n)$ is a terminal path in the d -graph, where $A_0 = A, r_0 = r, \alpha_1 \cdots \alpha_n = \alpha$, and $A_n r_n = u$. Assume that there is no i ($1 \leq i \leq m$) in the 0-segment $(A_0, r_0) d^\epsilon (A_1, r_1) \cdots d^\epsilon (A_m, r_m)$ of h such that $A_i = A$ and $r_i = r$. Then h is said to be an $\langle A, r, \alpha, u \rangle$ -path.*

The following property can be obtained similarly to Property 4.1.

PROPERTY 4.3. *Let $r \in R$. If there exists an $\langle A, r, \alpha, u \rangle$ -path in the d -graph, then there exists a derivation $Ar \Rightarrow_{A,R}^* \beta Bzr \Rightarrow_{A,R} \beta \gamma \delta zr \Rightarrow_{A,R}^* \beta \gamma yzr$ in G , where $\beta \gamma = \alpha$ and $k:yzr = u$, and vice versa.*

On the other hand, $RC_k^{A,R}$ can be obtained by examining some related paths in the d -graph as follows.

PROPERTY 4.4. $RC_k^{A,R}(\alpha) = \{u \mid \text{there exists an } \langle A, r, \alpha, u \rangle\text{-path in the } d\text{-graph where } r \in R\}$.

Proof. (\supseteq) Assume that there exists an $\langle A, r, \alpha, u \rangle$ -path in the d -graph, $h = (A, r) d^{*\alpha} (a, v)$, $r \in R$, where $av = u$. Then by Property 4.3, there exists a derivation $Ar \Rightarrow_{A,R}^* \beta Bzr \Rightarrow_{A,R} \beta \gamma \delta zr \Rightarrow_{A,R}^* \beta \gamma yzr$ in G , where $\beta \gamma = \alpha$ and $k:yzr = u$. Hence, $u \in RC_k^{A,R}(\alpha)$.

(\subseteq) Let $u \in RC_k^{A,R}(\alpha)$. Then there exists a derivation $Ar \Rightarrow_{A,R}^* \beta Bzr \Rightarrow_{A,R} \beta \gamma \delta zr \Rightarrow_{A,R}^* \beta \gamma yzr$ in G , where $\beta \gamma = \alpha$ and $k:yzr = u$ for some $r \in R$. By Property 4.3, there exists an $\langle A, r, \alpha, u \rangle$ -path in the d -graph. \square

4.2. $\mathbf{\Pi}$ relation. We define a predictive relation, written by $\mathbf{\Pi}$, which represents predictable symbols based on the generalized idea.

DEFINITION 4.2. *Let $A, B \in N, R \subseteq FOLLOW_k(A), W \subseteq FOLLOW_k(B), \beta, \gamma \in V^*$, where $(A, R, \beta \gamma) \neq (B, W, \gamma)$ and $u \in RC_k^{B,W}(\gamma)$. Then $(A, R, \beta \gamma) \mathbf{\Pi}_u (B, W, \gamma)$ iff, for each $\langle A, r, \beta \gamma, u \rangle$ -path, $r \in R$, we let $(A_0, r_0) d^\epsilon (A_1, r_1) \cdots (A_{n-1}, r_{n-1}) d^\epsilon (A_n, r_n)$ be the $|\beta|$ -segment of the path, then*

(i) *there exists m ($0 \leq m \leq n$) such that*

$$(4.3) \quad A_m = B$$

and

$$(4.4) \quad \bigcup_h \{r_i \mid A_i = B, 0 \leq i \leq m\} \cap \bigcup_h \{r_i \mid A_i = B, m+1 \leq i \leq n\} = \emptyset,$$

where h in (4.4) ranges over all $\langle A, r, \beta \gamma, u \rangle$ -paths, and

(ii) *if m_s is the smallest m ($m > 0$ when $\beta = \epsilon$) satisfying both (4.3) and (4.4), then*

$$(4.5) \quad W = \bigcup_h \{r_{m_s}\},$$

where h in (4.5) ranges over all $\langle A, r, \beta \gamma, u \rangle$ -paths.

$(A, R, \beta \gamma) \mathbf{\Pi}_u (B, W, \gamma)$ means that when the suffix $\beta \gamma$ of the current stack string is already predicted to be reduced to (A, R) ,¹ it is now predicted that whenever the k -length prefix of the remaining input is u , it is certain that the suffix γ of $\beta \gamma$ will be reduced to (B, W) . Figure 4.1 depicts this prediction; the left tree is expected to be the right one. This relation expresses the generalized prediction; compared to Hammer and Soisalon-Soininen's methods, γ is not restricted, and the prediction is performed over a predictive string u rather than over predictive language [3].

The smallest condition in Definition 4.2(ii) is meaningful only when B is left recursive. If B is not left recursive, then the m in Definition 4.2(i) is unique, and so the smallest condition becomes meaningless.

¹ (A, R) represents a refinement of A with the context information R such that (A, R) 's following string belongs to R and the generated language from (A, R) is $L(A, R) = \{x \mid Ar \Rightarrow_{A,R}^* xr, r \in R\}$.

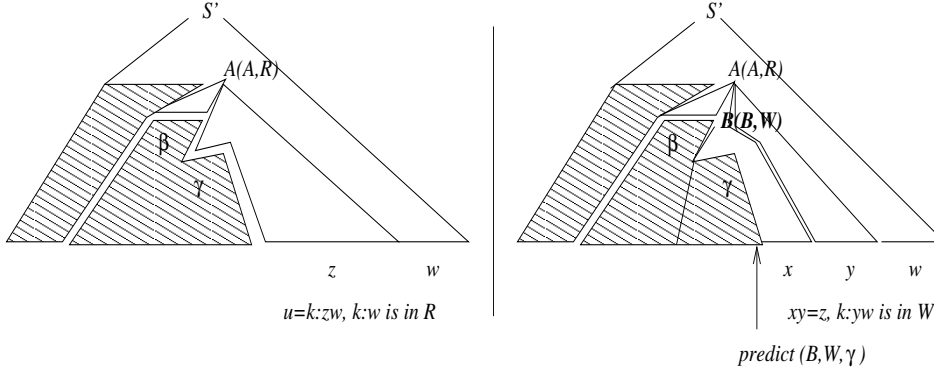


FIG. 4.1. Predictive property.

The following two lemmas delineate the meaning of $(A, R, \beta\gamma)\mathbf{II}_u(B, W, \gamma)$ in terms of grammatical derivation forms.

LEMMA 4.3. *Let $(A, R, \beta\gamma)\mathbf{II}_u(B, W, \gamma)$. Then whenever there exists $Ar \Rightarrow_{A,R}^* \beta\gamma zr$, where $r \in R$ and $k:zr=u$, there exist $Ar \Rightarrow_{A,R}^* \beta Byr$ and $Bw \Rightarrow_{B,W}^* \gamma xw$, where $w = k:yr (w \in W)$ and $xy = z$.*

Proof. Consider the derivation

$$\begin{aligned}
 (4.6) \quad & B_0 r \Rightarrow_{A,R} \beta_1 B_1 \delta_1 r \Rightarrow_{A,R}^* \beta_1 B_1 z_1 r \Rightarrow_{A,R} \beta_1 \beta_2 B_2 \delta_2 z_1 r \\
 & \Rightarrow_{A,R}^* \cdots \Rightarrow_{A,R}^* \beta_1 \beta_2 \cdots \beta_n B_n z_n \cdots z_1 r,
 \end{aligned}$$

where $B_0 = A$, $\beta_1 \cdots \beta_n = \beta\gamma$, and $k:zr = u$ with $B_n z_n \cdots z_1 = z$. Then there exists a corresponding path $h = (B_0, r_0) d^{\beta_1} (B_1, r_1) d^{\beta_2} (B_2, r_2) \cdots (B_{n-1}, r_{n-1}) d^{\beta_n} (B_n, r_n)$, where $r_0 = r$, $r_i = k:z_i \cdots z_1 r$ ($i = 1, \dots, n-1$), and $B_n r_n = u$. Here, h is an $\langle A, r, \beta\gamma, u \rangle$ -path. Let $(A_0, r_0) d^\epsilon (A_1, r_1) \cdots d^\epsilon (A_n, r_n)$ be the $|\beta|$ -segment of path h . By Definition 4.2 and (4.3), there exists A_m for some $m (0 \leq m \leq n)$ such that $A_m = B$, and by Definition 4.2 and (4.4), there is no $r_j \in W, m < j \leq n$. Thus, the derivation (4.6) can be divided into $Ar \Rightarrow_{A,R}^* \beta Byr$, where $\beta = \beta_1 \cdots \beta_m$ and $y = z_m \cdots z_1$ and $Bw \Rightarrow_{B,W}^* \gamma xw$, where $\gamma = \beta_{m+1} \cdots \beta_n$, $x = B_n z_n \cdots z_{m+1}$, and $w = k:yr = r_m (w \in W)$. \square

Similar to the proof of Lemma 4.3, we can prove the following lemma; the details are omitted.

LEMMA 4.4. *Let $A, B \in N, R \subseteq FOLLOW_k(A), W \subseteq FOLLOW_k(B), \beta, \gamma \in V^*$, and $u \in RC_k^{B,W}(\gamma)$. If, whenever there exists $Ar \Rightarrow_{A,R}^* \beta\gamma zr$ where $r \in R$ and $k:zr=u$, there exist $Ar \Rightarrow_{A,R}^* \beta Byr$ and $Bw \Rightarrow_{B,W}^* \gamma xw$ where $w = k:yr (w \in W)$ and $xy = z$, then $(A, R, \beta\gamma)\mathbf{II}_u(B, W', \gamma)$ for some W' .*

Let $(A, R, \beta\gamma)\mathbf{II}_u(B, W, \gamma)$ and $(A, R, \beta\gamma)\mathbf{II}_u(B, W', \gamma)$. Then W is equal to W' because of the smallest property in Definition 4.2 (ii).

The following example shows the computing process of \mathbf{II} relations.

Example 4.1. Let $G2 = (\{S, A, C, B, X, Y\}, \{a, b, c\}, P2, S)$, where $P2 = \{S \rightarrow A, S \rightarrow C, A \rightarrow BX, A \rightarrow BY, C \rightarrow Ba, B \rightarrow b, X \rightarrow BA, X \rightarrow c, Y \rightarrow BC\}$. Note that $G2$ is LR(1). See $\langle A, \$, BBB, b \rangle$ -paths and $\langle A, \$, BBB, c \rangle$ -paths in Figure 4.2.

(i) We can find two $\langle A, \$, BBB, b \rangle$ -paths, $p1 = (A, \$) d^B(X, \$) d^B(A, \$) d^B(X, \$) d^\epsilon(B, b) d^\epsilon(b, \epsilon)$ and $p2 = (A, \$) d^B(X, \$) d^B(A, \$) d^B(Y, \$) d^\epsilon(B, b) d^\epsilon(b, \epsilon)$. Then $p1$ and $p2$ have the same 2-segment $(A, \$)$. Hence, we have $(A, \{\$, \}, BBB) \mathbf{II}_b(A, \{\$, \}, B)$. On the other hand, $p1$ and $p2$ have the 3-segments $(X, \$) d^\epsilon(B, b) d^\epsilon(b, \epsilon)$ and $(Y, \$) d^\epsilon(B, b) d^\epsilon(b, \epsilon)$, respectively. Hence, we have $(A, \{\$, \}, BBB) \mathbf{II}_b(B, \{b\}, \epsilon)$.

(ii) We next find one $\langle A, \$, BBB, c \rangle$ -path, $p3 = (A, \$) d^B(X, \$) d^B(A, \$) d^B(X, \$) d^\epsilon(c, \epsilon)$. By examining this path, we have $(A, \{\$, \}, BBB) \mathbf{II}_c(A, \{\$, \}, B)$ and $(A, \{\$, \}, BBB) \mathbf{II}_c(X, \{\$, \}, \epsilon)$.

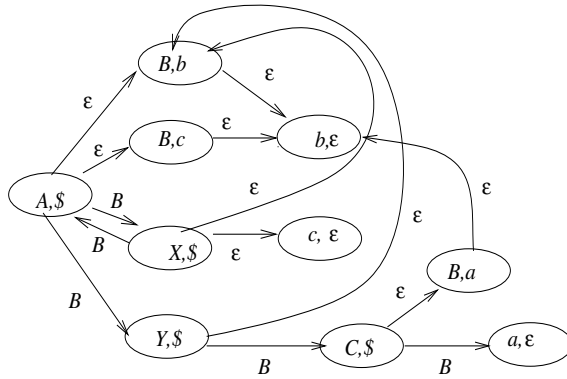


FIG. 4.2. A part of the d -graph.

In the computation of the \mathbf{II} relation, visiting vertices in the d -graph a bounded number of times is enough even when cycles are present in the d -graph.

5. Deterministic prediction. This section develops a deterministic procedure for choosing only one relation when there exist more than one \mathbf{II}_u relation with a fixed left side. In Hammer’s method, a heuristic is used for the selection. In this paper, the choice problem is resolved by choosing the nearer one in the derivation tree between the corresponding positions of (B, W, γ) and (C, X, ζ) to that of (A, R, α) . Let $\alpha = \beta\gamma$ and $\gamma = \delta\zeta$. The roots (A, R) , (B, W) , and (C, X) of the subtrees constructed from (A, R, α) , (B, W, γ) , and (C, X, ζ) are shown in Figure 5.1. In this situation, (B, W, γ) is selected since (A, R) is nearer to (B, W) than it is to (C, X) .

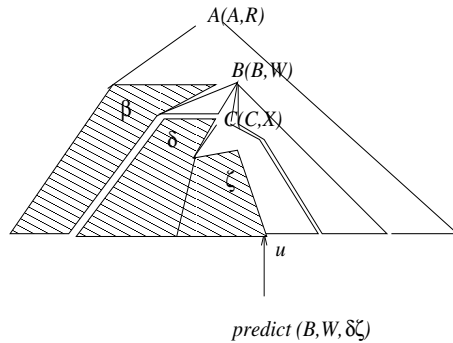


FIG. 5.1. Derivation tree.

In the remaining part of the paper, if $(A, R, \beta\gamma)\mathbf{II}_u(B, W, \gamma)$, then we assume $u \in \{k:x|Bw \Rightarrow_{B,W}^* \gamma xw, w \in W\}$ for some technical simplification.

The following lemma is used to show the orderable property of \mathbf{II} in the nearest sequence.

LEMMA 5.1.

(i) Let $(A, R, \beta\delta\zeta)\mathbf{II}_u(B, W, \delta\zeta)$ and $(A, R, \beta\delta\zeta)\mathbf{II}_u(C, X, \zeta)$. Then for each $w \in W$, there exists $x \in X$ such that $(B, w)d^{*\delta}(C, x)$.

(ii) Let $(A, R, \beta\delta\zeta)\mathbf{II}_u(B, W, \delta\zeta)$ and $(B, W, \delta\zeta)\mathbf{II}_u(C, X, \zeta)$. Then $(A, R, \beta\delta\zeta)\mathbf{II}_u(C, X, \zeta)$.

(iii) Let $(A, R, \beta\delta\zeta)\mathbf{II}_u(B, W, \delta\zeta)$ and $(A, R, \beta\delta\zeta)\mathbf{II}_u(C, X, \zeta)$, where $(B, W, \delta\zeta) \neq (C, X, \zeta)$. Then when $\delta \neq \epsilon$, $(B, W, \delta\zeta)\mathbf{II}_u(C, X, \zeta)$; when $\delta = \epsilon$, $(B, W, \zeta)\mathbf{II}_u(C, X, \zeta)$ or $(C, X, \zeta)\mathbf{II}_u(B, W, \zeta)$.

(iv) There are no $A, R, \alpha, u, B, W, \gamma$ such that $(A, R, \alpha)\mathbf{II}_u(B, W, \gamma)$ and $(B, W, \gamma)\mathbf{II}_u(A, R, \alpha)$, where $(A, R, \alpha) \neq (B, W, \gamma)$.

Proof.

(i) Let $w \in W$. Then there exists an $\langle A, r, \beta\delta\zeta, u \rangle$ -path h , where $r \in R$ which contains (B, w) on the $|\beta|$ -segment. The path h also contains (C, x) for some $x \in X$ on the $|\beta\delta|$ -segment. Hence, $(B, w)d^{*\delta}(C, x)$ holds, and Lemma 5.1(i) is true.

(ii) Let $r \in R$. Take an arbitrary $\langle A, r, \beta\delta\zeta, u \rangle$ -path h . Since $(A, R, \beta\delta\zeta)\mathbf{II}_u(B, W, \delta\zeta)$ holds, h contains (B, w) for some $w \in W$ on the $|\beta|$ -segment. Take an arbitrary $\langle B, w, \delta\zeta, u \rangle$ -path h_2 that is a suffix of h . Since $(B, W, \delta\zeta)\mathbf{II}_u(C, X, \zeta)$ holds, h_2 contains (C, x) for some $x \in X$ on the $|\delta|$ -segment. Hence, $(A, R, \beta\delta\zeta)\mathbf{II}_u(C, X', \zeta)$ holds. The next goal is to show $X' = X$. When $\delta \neq \epsilon$, it is obvious that $X' = X$. We next consider the case of $\delta = \epsilon$. Figure 5.2(a)–(c) shows the possible forms of an $\langle A, r, \beta\delta\zeta, u \rangle$ -path containing (B, w) and (C, x) on the $|\beta|$ -segment. By investigating the figure, we know that there are some paths in (a) and (c) that do not contain (B, w) or (C, x) , and so (b) is the only correct form. Hence, we can conclude that every (C, x) , $x \in X$ on the $|\beta|$ -segment of h follows (B, w) for some $w \in W$ on the same segment. As a consequence, X' has to be equal to X .

(iii) We first think about the case of $\delta \neq \epsilon$. Let $w \in W$. Take an arbitrary $\langle B, w, \delta\zeta, u \rangle$ -path h . Then according to Lemma 5.1(i), h has a $\langle C, x, \zeta, u \rangle$ -path for some $x \in X$ as a suffix. Hence, $(B, W, \delta\zeta)\mathbf{II}_u(C, X, \zeta)$ holds. Let us consider the other case of $\delta = \epsilon$. Then either one of $(B, W, \zeta)\mathbf{II}_u(C, X', \zeta)$ or $(C, X, \zeta)\mathbf{II}_u(B, W', \zeta)$ holds. Without loss of generality, we assume $(B, W, \zeta)\mathbf{II}_u(C, X', \zeta)$. At this point, by Lemma 5.1(ii), $(A, R, \beta\delta\zeta)\mathbf{II}_u(C, X', \zeta)$ holds. On the other hand, X' is uniquely determined, and thus $X' = X$. In all, $(B, W, \zeta)\mathbf{II}_u(C, X, \zeta)$ holds.

(iv) Assume that $(A, R, \alpha)\mathbf{II}_u(B, W, \gamma)$ and $(B, W, \gamma)\mathbf{II}_u(A, R, \alpha)$, where $(A, R, \alpha) \neq (B, W, \gamma)$. Then $\alpha = \gamma$, and $(A, R) \neq (B, W)$ because of $(A, R, \alpha) \neq (B, W, \gamma)$. Take an arbitrary $\langle B, w, \gamma, u \rangle$ -path h , $w \in W$. Then h is a suffix of an $\langle A, r, \alpha, u \rangle$ -path for some $r \in R$ because of $(A, R, \alpha)\mathbf{II}_u(B, W, \gamma)$. Next, take an arbitrary $\langle A, r, \alpha, u \rangle$ -path h_2 ; then, h_2 is a suffix of a $\langle B, w, \gamma, u \rangle$ -path for some $w \in W$ because of $(B, W, \gamma)\mathbf{II}_u(A, R, \alpha)$. However, the 0-segment of an $\langle A, r, \alpha, u \rangle$ -path cannot contain (A, r) in any other position but the first. As a result, both $(A, R, \alpha)\mathbf{II}_u(B, W, \gamma)$ and $(B, W, \gamma)\mathbf{II}_u(A, R, \alpha)$ cannot be true. \square

Let $(A, R, \alpha)\mathbf{II}_u(B, W, \gamma)$ and $(A, R, \alpha)\mathbf{II}_u(C, X, \zeta)$. Then by Lemma 5.1(iii), we can assume $(B, W, \gamma)\mathbf{II}_u(C, X, \zeta)$ without loss of generality, and then, by Lemma 5.1(iv), $(C, X, \zeta)\mathbf{II}_u(B, W, \gamma)$ does not hold. By generalizing this property, we have the following lemma.

LEMMA 5.2. Let $(A, R, \alpha)\mathbf{II}_u(B_0, W_0, \gamma_0)$, $(A, R, \alpha)\mathbf{II}_u(B_1, W_1, \gamma_1)$, \dots , (A, R, α)

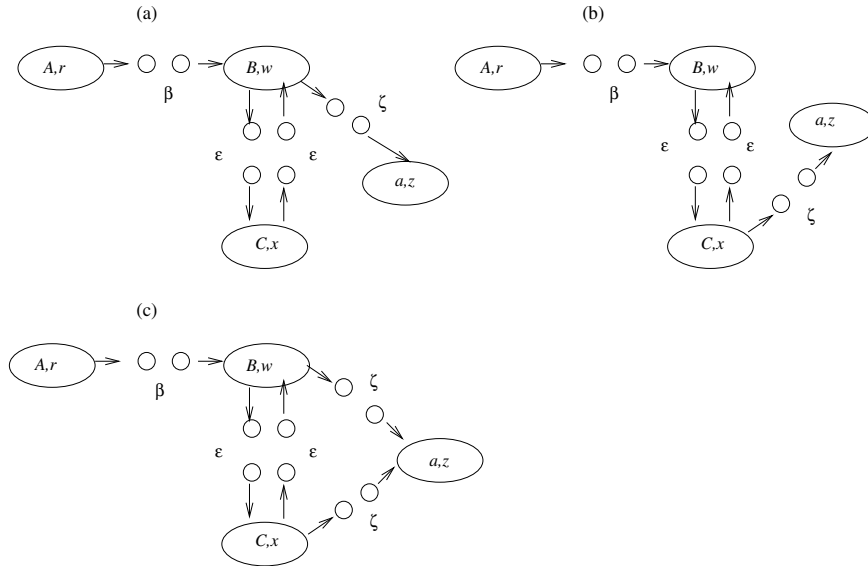


FIG. 5.2. Paths in the d -graph, where $az = u$.

$\mathbb{I}_u(B_n, W_n, \gamma_n)$ be distinct relations. For the elements $(B_0, W_0, \gamma_0), (B_1, W_1, \gamma_1), \dots, (B_n, W_n, \gamma_n)$, there exists a unique sequence $(B_{m_0}, W_{m_0}, \gamma_{m_0}), (B_{m_1}, W_{m_1}, \gamma_{m_1}), \dots, (B_{m_n}, W_{m_n}, \gamma_{m_n})$ such that $(B_{m_{i-1}}, W_{m_{i-1}}, \gamma_{m_{i-1}}) \mathbb{I}_u(B_{m_i}, W_{m_i}, \gamma_{m_i}), i = 1, \dots, n$.

The unique ordering property in Lemma 5.2 enable us to linearize \mathbb{I} relations. Consequently, we can define Φ function as follows.

DEFINITION 5.3. Let A, R, α, u be given. For all the distinct elements $(B_0, W_0, \gamma_0), (B_1, W_1, \gamma_1), \dots, (B_n, W_n, \gamma_n)$ such that $(A, R, \alpha) \mathbb{I}_u(B_i, W_i, \gamma_i), i = 0, \dots, n$, let $(B_{m_0}, W_{m_0}, \gamma_{m_0}), (B_{m_1}, W_{m_1}, \gamma_{m_1}), \dots, (B_{m_n}, W_{m_n}, \gamma_{m_n})$ be the sequence such that $(B_{m_{i-1}}, W_{m_{i-1}}, \gamma_{m_{i-1}}) \mathbb{I}_u(B_{m_i}, W_{m_i}, \gamma_{m_i}), i = 1, \dots, n$. Then $\Phi(A, R, \alpha, u)$ is defined as $(B_{m_0}, W_{m_0}, \gamma_{m_0})$; otherwise, $\Phi(A, R, \alpha, u)$ is undefined.

We have, in Example 4.1, $(A, \{\$, \}, BBB) \mathbb{I}_b(A, \{\$, \}, B)$ and $(A, \{\$, \}, BBB) \mathbb{I}_b(B, \{\$, \}, \epsilon)$. Then we have the unique sequence $(A, \{\$, \}, B), (B, \{\$, \}, \epsilon)$ satisfying the condition in Lemma 5.2; note that $(A, \{\$, \}, B) \mathbb{I}_b(B, \{\$, \}, \epsilon)$ holds. Hence, $\Phi(A, \{\$, \}, BBB, b)$ is defined as $(A, \{\$, \}, B)$. Similarly, $\Phi(A, \{\$, \}, BBB, c)$ is defined as $(A, \{\$, \}, B)$.

6. Extended PLR(k) grammars. Using the Φ function, we give a transformation into LL(k) form, and the transformable grammars are defined as extended PLR(k) grammars.

6.1. A transformation. Given G , the following algorithm is applied to obtain $T(G)$.

ALGORITHM 1 (construction of $T(G)$).

INPUT: G

OUTPUT: If this algorithm successfully terminates, $T(G) = (\mathbf{N}, \Sigma, \mathbf{P}, \mathbf{S})$ is constructed.

METHOD:

1. Initially,

$\mathbf{S} = [S, \{\$^k\}, \epsilon, \text{FIRST}_k(S\$^k)]; \mathbf{N} = \{\mathbf{S}\}; \mathbf{P} = \emptyset.$

2. **repeat**

(a) For each $[A, R, \alpha, U] \in \mathbf{N}$, let $Z = \{u \in U \mid \Phi(A, R, \alpha, u) \text{ is defined}\}$, then do the following:

Type 1. $\mathbf{P} = \mathbf{P} \cup \{[A, R, \alpha, U] \rightarrow a[A, R, \alpha a, V]\}$ if $V \neq \emptyset$, where $V = \{k:zwr \mid \text{there exists } Ar \Rightarrow_{A,R}^* \beta Bwr \Rightarrow_{A,R} \beta \gamma a \delta wr \Rightarrow_{A,R}^* \beta \gamma a zwr \text{ in } G, \text{ where } r \in R, \beta \gamma = \alpha, k:azwr \in U, \text{ and } k:azwr \notin Z\}.$

Type 2. $\mathbf{P} = \mathbf{P} \cup \{[A, R, \alpha, U] \rightarrow [A, R, \beta B, V]\}$, where $\alpha = \beta \gamma$ if $V \neq \emptyset$, where $V = \{k:wr \mid \text{there exists } Ar \Rightarrow_{A,R}^* \beta Bwr \Rightarrow_{A,R} \beta \gamma wr \text{ in } G, \text{ where } r \in R, k:wr \in U, \text{ and } k:wr \notin Z\}.$

Type 3. $\mathbf{P} = \mathbf{P} \cup \{[A, R, A, U] \rightarrow \epsilon\}$, where $R \cap U \neq \emptyset$ and $\alpha = A$.

Type 4. $\mathbf{P} = \mathbf{P} \cup \{[A, R, \alpha, U] \rightarrow [B, W, \gamma, V][A, R, \beta B, W]\}$, where $\alpha = \beta \gamma$ if $V \neq \emptyset$, where $V = \{u \mid \Phi(A, R, \alpha, u) = (B, W, \gamma), u \in U\}.$

(b) New nonterminals that appeared in \mathbf{P} are added to \mathbf{N} .

until (\mathbf{P} is not changed)

Algorithm 1 does not successfully terminate when an infinite number of nonterminals is generated. The formal characterization of the successfully transformable grammars is given in section 6.3. The remarkable observation is that our transformation depends on only G while Hammer's transformation depends on both G and a cycle-free multiple stack machine for G . As a result, given G , $\mathbf{T}(G)$ is constructed in a single process.

6.2. The LL(k) covering property. A homomorphism h is defined from \mathbf{P} to $\mathbf{P} \cup \{\epsilon\}$:

$$h(p_T) = \begin{cases} B \rightarrow \gamma & \text{if } p_T = [A, R, \beta \gamma, U] \rightarrow [A, R, \beta B, V], \\ \epsilon & \text{otherwise.} \end{cases}$$

The following two lemmas show a relationship between a rightmost derivation in G and a leftmost derivation in $\mathbf{T}(G)$.

LEMMA 6.1. Let $[A, R, \alpha, U] \in \mathbf{N}$. Assume that there exists π such that $Ar \Rightarrow_{A,R}^\pi \alpha xr$ in G , where $r \in R$ and $k:xr \in U$. Then there exists π_T such that $[A, R, \alpha, U] \Rightarrow_{lm}^{\pi_T} x$ in $\mathbf{T}(G)$, where $h(\pi_T) = \pi^R$.

Proof. We use induction on $|\pi|$. As the basis, assume that $|\pi| = 1$. Then $A \rightarrow \alpha x \in P$, and there exist an $|x|$ -length string π_T of Type 1 rules such that $[A, R, \alpha, U] \Rightarrow_{lm}^{\pi_T} x[A, R, \alpha x, T]$ and a rule string $p_T^1 p_T^2$ such that $[A, R, \alpha x, T] \Rightarrow_{lm}^{p_T^1} [A, R, A, M] \Rightarrow_{lm}^{p_T^2} \epsilon$ in $\mathbf{T}(G)$ for some M . Here, $h(\pi_T p_T^1 p_T^2) = h(\pi_T) h(p_T^1) h(p_T^2) = h(p_T^1) = A \rightarrow \alpha x$ because of $h(\pi_T) = \epsilon$ and $h(p_T^2) = \epsilon$. Hence, this lemma holds for the basis step. As an inductive hypothesis, assume that this lemma holds when $|\pi| \leq l (l > 1)$. Now suppose that $|\pi| > l$.

Case 1. There exist y, B, W , and γ such that $\Phi(A, R, \alpha y, u) = (B, W, \gamma)$, where $u = k:zr$ with $yz = x$.

In this case, we have a $|y|$ -length string π'_T of Type 1 rules such that $[A, R, \alpha, U] \Rightarrow_{lm}^{\pi'_T} y[A, R, \alpha y, X]$, where $u \in X$ and a rule p_T such that $[A, R, \alpha y, X] \Rightarrow_{lm}^{p_T} [B, W, \gamma, V][A, R, \beta B, W]$, where $u \in V$ and $\alpha y = \beta \gamma$ in $\mathbf{T}(G)$. On the other hand, π such that $Ar \Rightarrow_{A,R}^\pi \alpha yzr$ can be divided into π'' and π''' such that $Ar \Rightarrow_{A,R}^{\pi''} \beta Bz_2r$ and $Bw \Rightarrow_{B,W}^{\pi'''} \gamma z_1w$, where $w = k:z_2r (w \in W)$ and $z_1z_2 = z$ by Lemma 4.3. Note that $k:z_1w \in V$ and $k:z_2r \in W$. By applying the inductive hypothesis to π'' and π''' , we have $[A, R, \beta B, W] \Rightarrow_{lm}^{\pi''} z_2$ and $[B, W, \gamma, V] \Rightarrow_{lm}^{\pi'''} z_1$ in $\mathbf{T}(G)$, where $h(\pi''_T) = (\pi'')^R$ and $h(\pi'''_T) = (\pi''')^R$. Therefore, there exists a rule string $\pi_T (= \pi'_T p_T \pi''_T \pi'''_T)$ such

that $[A, R, \alpha, U] \Rightarrow_{lm}^{\pi_T} yz (= x)$ in $\mathbf{T}(G)$. Here, $h(\pi_T) = h(\pi'_T p_T \pi''_T \pi'''_T) = h(\pi''_T)h(\pi'''_T) = (\pi''_T)^R (\pi'''_T)^R = (\pi''_T \pi'''_T)^R = \pi^R$ because of $h(\pi'_T) = \epsilon$ and $h(p_T) = \epsilon$.

Case 2. The other cases.

(Subcase 2-1) The π is composed of π' and p such that $Ar \Rightarrow_{A,R}^{\pi'} \beta Bzr \Rightarrow_{A,R}^p \beta \gamma yzr$, where $\beta \gamma = \alpha$ and $yz = x$. Then there exist a $|y|$ -length string π''_T of Type 1 rules such that $[A, R, \alpha, U] \Rightarrow_{lm}^{\pi''_T} y[A, R, \alpha y, M]$, where $k:zr \in M$, and a Type 2 rule p_T such that $[A, R, \alpha y, M] \Rightarrow_{lm}^{p_T} [A, R, \beta B, W]$, where $k:zr \in W$ in $\mathbf{T}(G)$ according to the construction of $\mathbf{T}(G)$. On the other hand, by applying the inductive hypothesis to π' , we have $[A, R, \beta B, W] \Rightarrow_{lm}^{\pi'_T} z$ in $\mathbf{T}(G)$, where $h(\pi'_T) = (\pi')^R$. In all, there exists $\pi_T (= \pi''_T p_T \pi'_T)$ such that $[A, R, \alpha, U] \Rightarrow_{lm}^{\pi_T} yz$ in $\mathbf{T}(G)$, where $yz = x$. Here, $h(\pi_T) = h(\pi''_T p_T \pi'_T) = h(\pi''_T)h(p_T)h(\pi'_T) = h(p_T)h(\pi'_T) = ph(\pi'_T) = p\pi'^R = (\pi'p)^R = \pi^R$ because of $h(\pi''_T) = \epsilon$.

(Subcase 2-2) The π is composed of π' and p such that $Ar \Rightarrow_{A,R}^{\pi'} \alpha y_1 Bzr \Rightarrow_{A,R}^p \alpha y_1 y_2 zr$, where $y_1 y_2 z = x$. Then there exist a $|y_1 y_2|$ -length string π''_T of Type 1 rules such that $[A, R, \alpha, U] \Rightarrow_{lm}^{\pi''_T} y_1 y_2 [A, R, \alpha y_1 y_2, M]$, where $k:zr \in M$, and a Type 2 rule p_T such that $[A, R, \alpha y_1 y_2, M] \Rightarrow_{lm}^{p_T} [A, R, \alpha y_1 B, W]$, where $k:zr \in W$ in $\mathbf{T}(G)$. On the other hand, we have $[A, R, \alpha y_1 B, W] \Rightarrow_{lm}^{\pi'_T} z$ in $\mathbf{T}(G)$, where $h(\pi'_T) = (\pi')^R$ by applying the inductive hypothesis to π' . In all, there exists $\pi_T (= \pi''_T p_T \pi'_T)$ such that $[A, R, \alpha, U] \Rightarrow_{lm}^{\pi_T} y_1 y_2 z$ in $\mathbf{T}(G)$. Here, $h(\pi_T) = h(\pi''_T p_T \pi'_T) = h(\pi''_T)h(p_T)h(\pi'_T) = h(p_T)h(\pi'_T) = ph(\pi'_T) = p(\pi')^R = (\pi'p)^R = \pi^R$ because of $h(\pi''_T) = \epsilon$. \square

LEMMA 6.2. *If there exists π_T such that $[A, R, \alpha, U] \Rightarrow_{lm}^{\pi_T} x$ in $\mathbf{T}(G)$, then there exists $Ar \Rightarrow_{A,R}^{h(\pi_T)^R} \alpha xr$ in G , where $r \in R$ and $k:xr \in U$.*

Proof. If $|\pi_T| = 1$, then $x = \epsilon, \alpha = A$, and $R \cap U \neq \emptyset$. Let $r \in R \cap U$; then there exists $Ar \Rightarrow_{A,R}^\epsilon Ar$ in G . Note that $k:xr \in U$ because of $k:xr = k:r = r$ and $r \in U$; $h(\pi_T)^R = \epsilon^R = \epsilon$. We have completed the proof of the basis case. Next, as an inductive hypothesis, assume that this lemma holds when $|\pi_T| \leq l (l > 1)$. Now let $|\pi_T| > l$ and $\pi_T = p_T \pi'_T$. Then p_T is a rule of Type 1, Type 2, or Type 4. According to the type, we divide the cases as follows.

Case 1. $p_T = [A, R, \alpha, U] \rightarrow a[A, R, \alpha a, Q]$.

Let $x = az$. Then there exists π'_T such that $[A, R, \alpha a, Q] \Rightarrow_{lm}^{\pi'_T} z$ in $\mathbf{T}(G)$. By applying the inductive hypothesis to π'_T , we have $Ar \Rightarrow_{A,R}^{h(\pi'_T)^R} \alpha azr$ in G , where $r \in R$ and $k:zr \in Q$. According to the construction of the set Q , $k:azr \in U$. Note that $h(\pi_T)^R = h(p_T \pi'_T)^R = h(\pi'_T)^R$ because of $h(p_T) = \epsilon$.

Case 2. $p_T = [A, R, \alpha, U] \rightarrow [A, R, \beta B, W]$.

Let $\alpha = \beta \gamma$ and $p = B \rightarrow \gamma$. Then there exists π'_T such that $[A, R, \beta B, W] \Rightarrow_{lm}^{\pi'_T} x$ in $\mathbf{T}(G)$, and p is a rule in P . By applying the inductive hypothesis to π'_T , we have $Ar \Rightarrow_{A,R}^{h(\pi'_T)^R} \beta Bxr$ in G , where $r \in R$ and $k:xr \in W$. On the other hand, the condition of $k:xr \in U$ is true according to $\mathbf{T}(G)$'s construction. Hence, we have the derivation $Ar \Rightarrow_{A,R}^{h(\pi'_T)^R} \beta Bxr \Rightarrow_{A,R}^p \beta \gamma xr$ in G , where $r \in R$ and $k:xr \in U$. Here, $h(\pi'_T)^R p = (ph(\pi'_T))^R = (h(p_T)h(\pi'_T))^R = h(p_T \pi'_T)^R = h(\pi_T)^R$.

Case 3. $p_T = [A, R, \alpha, U] \rightarrow [B, W, \gamma, Y][A, R, \beta B, W]$.

Let $[B, W, \gamma, Y] \Rightarrow_{lm}^{\pi''_T} x_1$ and $[A, R, \beta B, W] \Rightarrow_{lm}^{\pi'''_T} x_2$ in $\mathbf{T}(G)$, where $x_1 x_2 = x$ and $p_T \pi''_T \pi'''_T = \pi_T$. Then by applying the inductive hypothesis to π''_T and π'''_T , we have $Bw \Rightarrow_{B,W}^{h(\pi''_T)^R} \gamma x_1 w$, where $w \in W$ and $k:x_1 w \in Y$, and $Ar \Rightarrow_{A,R}^{h(\pi'''_T)^R} \beta Bx_2 r$, where $r \in R$ and $k:x_2 r \in W$ in G . Here, $k:x_1 w \in U$ because of $Y \subseteq U$. Without loss of generality, we set w to be $k:x_2 r$. Then we have $Ar \Rightarrow_{A,R}^{h(\pi''_T)^R} \beta Bx_2 r \Rightarrow_{A,R}^{h(\pi'''_T)^R} \beta \gamma x_1 x_2 r$ in G . Here, $k:x_1 x_2 r = k:x_1 (k:x_2 r) = k:x_1 w$, and so $k:x_1 x_2 r \in U$. Note that

$h(\pi_T''')^R h(\pi_T'')^R = (h(\pi_T'')h(\pi_T'''))^R = (h(p_T)h(\pi_T'')h(\pi_T'''))^R = h(p_T\pi_T''\pi_T''')^R = h(\pi_T)^R$
because of $h(p_T) = \epsilon$.

In all, we have this lemma. \square

From Lemmas 6.1 and 6.2, we get the following corollaries.

COROLLARY 6.3. *If there exists π such that $S\mathbb{S}^k \Rightarrow_{rm}^\pi x\mathbb{S}^k$ in G , then there exists π_T such that $[S, \{\mathbb{S}^k\}, \epsilon, FIRST_k(S\mathbb{S}^k)] \Rightarrow_{lm}^{\pi_T} x$ in $\mathbf{T}(G)$, where $h(\pi_T) = \pi^R$.*

Proof. If there exists π such that $S\mathbb{S}^k \Rightarrow_{rm}^\pi x\mathbb{S}^k$ in G , then there exists $S\mathbb{S}^k \Rightarrow_{S, \{\mathbb{S}^k\}}^\pi x\mathbb{S}^k$ in G . On the other hand, $[S, \{\mathbb{S}^k\}, \epsilon, FIRST_k(S\mathbb{S}^k)] \in \mathbf{N}$. Then according to Lemma 6.1, there exists π_T such that $[S, \{\mathbb{S}^k\}, \epsilon, FIRST_k(S\mathbb{S}^k)] \Rightarrow_{lm}^{\pi_T} x$ in $\mathbf{T}(G)$, where $h(\pi_T) = \pi^R$. \square

COROLLARY 6.4. *If there exists π_T such that $[S, \{\mathbb{S}^k\}, \epsilon, FIRST_k(S\mathbb{S}^k)] \Rightarrow_{lm}^{\pi_T} x$ in $\mathbf{T}(G)$, then there exists $S\mathbb{S}^k \Rightarrow_{rm}^{h(\pi_T)^R} x\mathbb{S}^k$ in G .*

Proof. By Lemma 6.2, if there exists π_T such that $[S, \{\mathbb{S}^k\}, \epsilon, FIRST_k(S\mathbb{S}^k)] \Rightarrow_{lm}^{\pi_T} x$ in $\mathbf{T}(G)$, then there exists $S\mathbb{S}^k \Rightarrow_{S, \{\mathbb{S}^k\}}^{h(\pi_T)^R} x\mathbb{S}^k$ in G , and so there exists $S\mathbb{S}^k \Rightarrow_{rm}^{h(\pi_T)^R} x\mathbb{S}^k$ in G . \square

According to Corollaries 6.3 and 6.4, we conclude the following theorem.

THEOREM 6.5. *$\mathbf{T}(G)$ left-to-right covers G with respect to h .*

On the other hand, the LL property of $\mathbf{T}(G)$ can be proved as follows.

THEOREM 6.6. *$\mathbf{T}(G)$ is LL(k).*

Proof. Suppose that $\mathbf{T}(G)$ is not LL(k). Then there exist two derivations in $\mathbf{T}(G)$ such that

$$[S, \{\mathbb{S}^k\}, \epsilon, FIRST_k(S\mathbb{S}^k)] \Rightarrow_{lm}^* x[A, R, \alpha, U]_\tau \Rightarrow_{lm} x\omega_\tau \Rightarrow_{lm}^* xy_\tau \Rightarrow_{lm}^* xyz$$

and

$$[S, \{\mathbb{S}^k\}, \epsilon, FIRST_k(S\mathbb{S}^k)] \Rightarrow_{lm}^* x[A, R, \alpha, U]_{\tau'} \Rightarrow_{lm} x\rho_{\tau'} \Rightarrow_{lm}^* xy'_{\tau'} \Rightarrow_{lm}^* xy'z',$$

where $k:yz = k:y'z'$ and $\omega \neq \rho$. Here, we can observe that $k:z\mathbb{S}^k \in R$ and $k:z'\mathbb{S}^k \in R$. Let r and r' denote $k:z\mathbb{S}^k$ and $k:z'\mathbb{S}^k$, respectively. At this time, we have $k:yr = k:y'r'$. Denote $[A, R, \alpha, U] \rightarrow \omega$ and $[A, R, \alpha, U] \rightarrow \rho$ by p_T^1 and p_T^2 , respectively. The possible situations for p_T^1 and p_T^2 are divided as follows.

Case 1. $p_T^1 = [A, R, \alpha, U] \rightarrow a[A, R, \alpha a, V]$.

Then there exists $[A, R, \alpha a, V] \Rightarrow_{lm}^* y_1$ in $\mathbf{T}(G)$, where $ay_1 = y$. By applying Lemma 6.2 to this derivation, we have

$$(6.1) \quad Ar \Rightarrow_{A,R}^* \alpha ay_1 r \text{ in } G.$$

Derivation (6.1) is of the form

$$(6.2) \quad Ar \Rightarrow_{A,R}^* \beta Bwr \Rightarrow_{A,R} \beta \gamma a \delta wr \Rightarrow_{A,R}^* \beta \gamma ay_1 r,$$

where $\beta\gamma = \alpha$ according to Type 1 rule's construction.

(Subcase 1-1) $p_T^2 = [A, R, \alpha, U] \rightarrow [A, R, \beta B, V]$.

Then there exists $[A, R, \beta B, V] \Rightarrow_{lm}^* y'$ in $\mathbf{T}(G)$. By applying Lemma 6.2 to this derivation, we have $Ar' \Rightarrow_{A,R}^* \beta By'r'$ in G , and if $\alpha = \beta\gamma$, then we know that there exists

$$(6.3) \quad Ar \Rightarrow_{A,R}^* \beta By'r \Rightarrow_{A,R} \beta \gamma y'r \text{ in } G.$$

At this point, we have an LR(k) conflict with the derivations (6.2) and (6.3) because of $k:ay_1r = k:y'r'$.

(Subcase 1-2) $p_T^2 = [A, R, A, U] \rightarrow \epsilon$.

In this case, $\alpha = A$ and $y' = \epsilon$. It implies that $k:yr$ is also in R because of $k:yr = k:y'r' = r'$ and $r' \in R$. However, $k:yr (= k:ay_1r)$ cannot be in R by the definition of $\Rightarrow_{A,R}$ because we already have the derivation (6.2).

(Subcase 1-3) $p_T^2 = [A, R, \alpha, U] \rightarrow [B, W, \gamma, V][A, R, \beta B, W]$, where $\beta\gamma = \alpha$.

Let $[B, W, \gamma, V] \Rightarrow_{lm}^* y'_1$ and $[A, R, \beta B, W] \Rightarrow_{lm}^* y'_2$ in $\mathbf{T}(G)$, where $y'_1 y'_2 = y'$. By applying Lemma 6.2 to these derivations, there exist $Bw \Rightarrow_{B,W}^* \gamma y'_1 w$, where $w \in W$ and $k:y'_1 w \in V$, and $Ar' \Rightarrow_{A,R}^* \beta B y'_2 r'$, where $k:y'_2 r' = w$ in G . Then we know that $\Phi(A, R, \alpha, k:y'_1 w) = (B, W, \gamma)$. This is a contradiction to $k:ay_1r = k:y'_1 y'_2 r'$ because $k:ay_1r$ cannot be in $\{u \in U \mid \Phi(A, R, \alpha, u) \text{ is defined}\}$ according to the construction of $\mathbf{T}(G)$.

Case 2. $p_T^1 = [A, R, \alpha, U] \rightarrow [A, R, \beta B, V]$.

Then there exists $[A, R, \beta B, V] \Rightarrow_{lm}^* y$ in $\mathbf{T}(G)$. By applying Lemma 6.2 to this derivation, we have $Ar \Rightarrow_{A,R}^* \beta B yr$ in G . If $\alpha = \beta\gamma$, then $B \rightarrow \gamma \in P$ and there exists

$$(6.4) \quad Ar \Rightarrow_{A,R}^* \beta B yr \Rightarrow_{A,R} \beta \gamma yr \text{ in } G.$$

(Subcase 2-1) $p_T^2 = [A, R, \alpha, U] \rightarrow [A, R, \zeta C, Z]$.

Then there exists $[A, R, \zeta C, Z] \Rightarrow_{lm}^* y'$ in $\mathbf{T}(G)$. By applying Lemma 6.2 to this derivation, there exists $Ar' \Rightarrow_{A,R}^* \zeta C y' r'$ in G , and if $\alpha = \zeta\delta$, then there exists

$$(6.5) \quad Ar' \Rightarrow_{A,R}^* \zeta C y' r' \Rightarrow_{A,R} \zeta \delta y' r' \text{ in } G.$$

At this time, we have an LR(k) conflict with the derivations (6.4) and (6.5) because of $B \rightarrow \gamma \neq C \rightarrow \delta$, although $\beta\gamma = \zeta\delta$ and $k:yr = k:y'r'$.

(Subcase 2-2) $p_T^2 = [A, R, A, U] \rightarrow \epsilon$.

In this case, $\alpha = A$ and $y' = \epsilon$. If $\beta = \epsilon$, then $\gamma = A$, and we have a contradiction to derivation (6.4) similar to (Subcase 1-2). If $\beta \neq \epsilon$, then $\beta = A$ and $\gamma = \epsilon$. Then there exists $Ar \Rightarrow_{A,R}^* AByr \Rightarrow_{A,R} Ayr$ in G . Here, $k:yr = k:y'r'$, and so $k:yr \in R$ because of $y' = \epsilon$ and $r' \in R$. However, it is a contradiction to the definition of $\Rightarrow_{A,R}$.

(Subcase 2-3) $p_T^2 = [A, R, \alpha, U] \rightarrow [C, X, \delta, Z][A, R, \zeta C, X]$, where $\zeta\delta = \alpha$.

In this case, we have a contradiction for the construction of $\mathbf{T}(G)$ in a similar way to (Subcase 1-3).

Case 3. $p_T^1 = [A, R, A, U] \rightarrow \epsilon$.

Then $\alpha = A$, $y = \epsilon$, and the only possible form of p_T^2 is $[A, R, \alpha, U] \rightarrow [B, W, \gamma, V][A, R, \beta B, W]$, where $\beta\gamma = \alpha$. Let $[B, W, \gamma, V] \Rightarrow_{lm}^* y'_1$ and $[A, R, \beta B, W] \Rightarrow_{lm}^* y'_2$ in $\mathbf{T}(G)$, where $y'_1 y'_2 = y'$. By applying Lemma 6.2 to these derivations, we have $Bw \Rightarrow_{B,W}^* \gamma y'_1 w$, where $w \in W$, and $Ar' \Rightarrow_{A,R}^* \beta B y'_2 r'$, where $k:y'_2 r' = w$ in G . Hence, there exists $Ar' \Rightarrow_{A,R}^* \beta B y'_2 r' \Rightarrow_{A,R}^* \beta \gamma y'_1 y'_2 r'$. Note that $\beta\gamma = A$ and $k:y'_1 y'_2 r' = r$ because of $k:y'_1 y'_2 r' = k:yr$ and $y = \epsilon$. It means that $k:y'_1 y'_2 r'$ is in R , but this containment cannot occur by the definition of $\Rightarrow_{A,R}$.

Case 4. $p_T^1 = [A, R, \alpha, U] \rightarrow [B, W, \gamma, V][A, R, \beta B, W]$, where $\beta\gamma = \alpha$.

Let $[B, W, \gamma, V] \Rightarrow_{lm}^* y_1$ and $[A, R, \beta B, W] \Rightarrow_{lm}^* y_2$ in $\mathbf{T}(G)$, where $y_1 y_2 = y$. By applying Lemma 6.2 to these derivations, we have $Bw \Rightarrow_{B,W}^* \gamma y_1 w$, where $w \in W$ and $Ar \Rightarrow_{A,R}^* \beta B y_2 r$, where $k:y_2 r = w$ in G . We note that $\Phi(A, R, \alpha, k:y_1 w) = (B, W, \gamma)$. On the other hand, the only possible form of p_T^2 is $[A, R, \alpha, U] \rightarrow [C, X, \delta, Z][A, R, \zeta C, X]$, where $\zeta\delta = \alpha$. Let $[C, X, \delta, Z] \Rightarrow_{lm}^* y'_1$ and $[A, R, \zeta C, X] \Rightarrow_{lm}^* y'_2$ in $\mathbf{T}(G)$, where $y'_1 y'_2 = y'$. By applying Lemma 6.2 to these derivations, we have $Cx' \Rightarrow_{C,X}^* \delta y'_1 x'$, where $x' \in X$ and $k:y'_1 x' \in Z$, and $Ar' \Rightarrow_{A,R}^* \zeta C y'_2 r'$, where $k:y'_2 r' = x'$ in G . Here,

$\Phi(A, R, \alpha, k:y_1'x') = (C, X, \delta)$. Then we have a contradiction to the definition of Φ function because of $k:y_1w = k:y_1'x'$.

For all the possible cases of p_T^1 and p_T^2 , we showed some contradictions, and hence $T(G)$ has to be LL(k). \square

From Theorems 6.5 and 6.6, we obtain that $T(G)$ is an LL(k) covering grammar of G .

6.3. Transformable grammars. We define some special nonterminals to detect the infinite process of Algorithm 1. Let $[A, R, \alpha, U] \in \mathbf{N}$ and $\alpha = X_1 \cdots X_n$. Assume that $q_i = \{[B \rightarrow \gamma.\delta, k:zr] \mid \text{there exists } Ar \Rightarrow_{A,R}^* \beta Bzr \Rightarrow_{A,R} \beta\gamma\delta zr \Rightarrow_{A,R}^* \beta\gamma\zeta yzr \text{ in } G, \text{ where } r \in R, k:yzr \in U, \beta\gamma = X_1 \cdots X_i, \text{ and } \zeta = X_{i+1} \cdots X_n\}$ for each $i = 0, 1, \dots, n$. For q_i, q_{i+1}, \dots, q_j , if $q_i = q_j (0 \leq i < j \leq n)$ and no other pair of q_i, q_{i+1}, \dots, q_j is identical, then q_i, \dots, q_j is a *loop*. We say that $[A, R, \alpha, U]$ is *cyclic* if there exist more than two different values for $i, 0 \leq i \leq n$ for the same loop such that q_i, \dots, q_j is a loop. If $[A, R, \alpha, U]$ is cyclic and there is no $u \in U$ such that $\Phi(A, R, \alpha, u) = (B, W, \gamma)$ for some B, W , and γ , then $[A, R, \alpha, U]$ is an *indivisible cyclic*. Using this nonterminal, we can characterize the transformable grammars as follows.

LEMMA 6.7. *Algorithm 1 with G successfully terminates iff no indivisible cyclic nonterminal is generated.*

Proof. (Only if part) Assume that an indivisible cyclic nonterminal $[A, R, \alpha, U]$ is generated during the execution of Algorithm 1 with G . Then an infinite number of cyclic nonterminals that have the common loop with $[A, R, \alpha, U]$ are generated according to Algorithm 1. Hence, Algorithm 1 does not successfully terminate.

(If part) During the execution of Algorithm 1, if no indivisible cyclic nonterminal is generated, then the length of each nonterminal generated is bounded. It means that Algorithm 1 successfully terminates. \square

As a result, whenever an indivisible cyclic nonterminal is found during the working of Algorithm 1, it is desirable to stop anymore processing of the algorithm.

Next we will give another characterization of the transformable grammars using grammatical derivations.

THEOREM 6.8. *Algorithm 1 with G successfully terminates iff (Statement 1) is true:*

(Statement 1) *There exists a constant n , depending on G , such that if $\alpha (\neq \epsilon)$ is a viable prefix of G , x is a string in $L^G(\alpha)$, and v is a string in $RC_k^G(\alpha)$, then there exist B, W , and $\gamma (\neq \epsilon)$, where $\alpha = \beta\gamma$ and $|\gamma| \leq n$ such that Condition 1 holds.*

(Condition 1) *Whenever there exists π such that $S' \Rightarrow_{r_m}^\pi \beta\gamma z \Rightarrow_{r_m}^* xz$ in G , where $k:z = v$, there exist π^1 and π^2 , where $\pi^1\pi^2 = \pi$ such that $S' \Rightarrow_{r_m}^{\pi^1} \beta Bz''$ and $Bw \Rightarrow_{B,W}^{\pi^2} \gamma z'w$ in G , where $w = k:z'' (w \in W)$ and $z'z'' = z$.*

Proof. (Only if part) Assume that Algorithm 1 with G successfully terminates, but there is no n satisfying Statement 1. In other words, there exist a viable prefix α of G , a string $x, x \in L^G(\alpha)$, and a string $v, v \in RC_k^G(\alpha)$ for which there are no predicted B, W , and γ satisfying Condition 1, where $\alpha = \beta\gamma$ and $|\gamma| \leq n$ for some fixed n . Take an arbitrary rule string $\hat{\pi} (\hat{\pi} = \pi'\pi'')$ such that there exists $S\$\$^k \Rightarrow_{r_m}^{\pi'} \alpha z \Rightarrow_{r_m}^{\pi''} xz$ in G , where $k:z = v$. Corresponding to $\hat{\pi} (= \pi'\pi'')$, there exists $\hat{\pi}_T$ such that $[S, \{\$\$^k\}, \epsilon, FIRST_k(S\$\$^k)] \Rightarrow_{l_m}^{\hat{\pi}_T} xz$ in $T(G)$, where $h(\hat{\pi}_T) = \hat{\pi}^R$ according to Lemma 6.1. Furthermore, we know that $\hat{\pi}_T$ is composed of π_T'' and π_T' such that $[S, \{\$\$^k\}, \epsilon, FIRST_k(S\$\$^k)] \Rightarrow_{l_m}^{\pi_T''} \bar{x}\tau \Rightarrow_{l_m}^{\pi_T'} \bar{x}\bar{z}$ in $T(G)$, $\tau \in \mathbf{N}^*$, where $\bar{x}\bar{z} = xz, h(\pi_T'')^R = \pi''$, and $h(\pi_T')^R = \pi'$.

Let τ be $[B_n, W_n, \gamma_n, U_n][B_{n-1}, W_{n-1}, \gamma_{n-1}B_n, W_n] \cdots [B_1, W_1, \gamma_1B_2, W_2]$. Suppose that $\pi_T^i = [B_i, W_i, \gamma_iB_{i+1}, W_{i+1}] \Rightarrow_{lm}^* z_i$ for each $i = 1, \dots, n-1$ and $\pi_T^n = [B_n, W_n, \gamma_n, U_n] \Rightarrow_{lm}^* z_n$, where $\pi_T^1 \cdots \pi_T^n = \pi_T'$ and $z_n \cdots z_1 = \bar{z}$. Then by applying Lemma 6.2 to π_T^1, \dots, π_T^n , respectively, we have $B_i w_i \Rightarrow_{B_i, W_i}^{h(\pi_T^i)^R} \gamma_i B_{i+1} z_i w_i$, where $w_i \in W_i$ and $k:z_i w_i \in W_{i+1}$ for each $i = 1, \dots, n-1$, and $B_n w_n \Rightarrow_{B_n, W_n}^{h(\pi_T^n)^R} \gamma_n z_n w_n$, where $w_n \in W_n$ and $k:z_n w_n \in U_n$ in G . Because of $\pi_T^n \pi_T^{n-1} \cdots \pi_T^1 = \pi_T'$, $h(\pi_T^1)^R \cdots h(\pi_T^n)^R = (h(\pi_T^n) \cdots h(\pi_T^1))^R = h(\pi_T^n \cdots \pi_T^1)^R = h(\pi_T')^R = \pi'$. Here, we have $z = \bar{z}$, and so $x = \bar{x}$; $\gamma_1 \cdots \gamma_n = \alpha$.

On the other hand, we know that π_T'' is of the form

$$\begin{aligned} & [S, \{\mathcal{S}^k\}, \epsilon, FIRST_k(S\mathcal{S}^k)] \\ & \Rightarrow_{lm}^{\pi_T^X} x' [B_{n-1}, W_{n-1}, \gamma_{n-1}\delta_n, X_{n-1}] \cdots [B_1, W_1, \gamma_1B_1, W_2] \\ & \Rightarrow_{lm}^{r_T} x' [B_n, W_n, \delta_n, V_n][B_{n-1}, W_{n-1}, \gamma_{n-1}B_n, W_n] \cdots [B_1, W_1, \gamma_1B_1, W_2] \\ & \Rightarrow_{lm}^{\pi_T^Y} x' x'' [B_n, W_n, \gamma_n, U_n][B_{n-1}, W_{n-1}, \gamma_{n-1}B_n, W_n] \cdots [B_1, W_1, \gamma_1B_1, W_2], \end{aligned}$$

where $x'x'' = x$. Hence, we have $\Phi(B_{n-1}, W_{n-1}, \gamma_{n-1}\delta_n, u) = (B_n, W_n, \delta_n)$, where $u = k:x''z$. Consequently, we have the following statement:

(Statement 2) whenever there exists $B_{n-1}w_{n-1} \Rightarrow_{B_{n-1}, W_{n-1}}^* \gamma_{n-1}\delta_n p w_{n-1}$, where $w_{n-1} \in W_{n-1}$ and $k:p w_{n-1} = u$, there exist $B_{n-1}w_{n-1} \Rightarrow_{B_{n-1}, W_{n-1}}^* \gamma_{n-1}B_n q w_{n-1}$ and $B_n w_n \Rightarrow_{B_n, W_n}^* \delta_n r w_n$, where $w_n = k:q w_{n-1}$ and $r q = p$. Additionally, we know that when α, x , and v are fixed, $B_1, W_1, \gamma_1, \dots, B_i, W_i, \gamma_i, \dots, B_n, W_n, \gamma_n, \delta_n, U_n$ are also fixed because $T(G)$ is LL(k).

Consider the derivation $\pi_T^s = r_T \pi_T^Y \pi_T^n \pi_T^{n-1}$ such that $[B_{n-1}, W_{n-1}, \gamma_{n-1}\delta_n, X_{n-1}] \Rightarrow_{lm}^{\pi_T^s} x'' z_n z_{n-1}$. Then by applying Lemma 6.2, there exists $h(\pi_T^s)^R$ such that $B_{n-1}w_{n-1} \Rightarrow_{B_{n-1}, W_{n-1}}^* \gamma_{n-1}\delta_n x'' z_n z_{n-1} w_{n-1}$ in G , where $w_{n-1} \in W_{n-1}$ and $k:x'' z_n z_{n-1} w_{n-1} \in X_{n-1}$. Here, $h(\pi_T^s)^R = h(r_T \pi_T^Y \pi_T^n \pi_T^{n-1})^R = h(\pi_T^Y \pi_T^n \pi_T^{n-1})^R = h(\pi_T^{n-1})^R h(\pi_T^n)^R h(\pi_T^Y)^R$ because of $h(r_T) = \epsilon$. Since π_T^Y is a suffix of π_T'' , $h(\pi_T^Y)^R$ is a prefix of π_T'' . Let π^Z be the rule string such that $\pi_T'' = h(\pi_T^Y)^R \pi^Z$. Then $\hat{\pi}$ is composed of the derivations such that

$$\begin{aligned} & B_1 w_1 \\ & \Rightarrow_{rm}^{h(\pi_T^1)^R h(\pi_T^2)^R \cdots h(\pi_T^{n-2})^R} \gamma_1 \gamma_2 \gamma_3 \cdots \gamma_{n-2} B_{n-1} z_{n-2} \cdots z_2 z_1 w_1 \\ & \Rightarrow_{rm}^{h(\pi_T^{n-1})^R} \gamma_1 \gamma_2 \gamma_3 \cdots \gamma_{n-1} B_n z_{n-1} \cdots z_2 z_1 w_1 \\ & \Rightarrow_{rm}^{h(\pi_T^n)^R} \gamma_1 \gamma_2 \cdots \gamma_{n-1} \gamma_n z_n z_{n-1} \cdots z_2 z_1 w_1 \\ & \Rightarrow_{rm}^{h(\pi_T^Y)^R} \gamma_1 \gamma_2 \cdots \gamma_{n-1} \delta_n x'' z_n z_{n-1} \cdots z_2 z_1 w_1 \\ & \Rightarrow_{rm}^{\pi^Z} x' x'' z_n z_{n-1} \cdots z_2 z_1 w_1, \end{aligned}$$

where $B_1 = S$, $w_1 = \mathcal{S}^k$, $\gamma_1 \cdots \gamma_n = \alpha$, and $z_n z_{n-1} \cdots z_2 z_1 = z$. Then we have that whenever there exists π such that $S' \Rightarrow_{rm}^{\pi} \alpha z \Rightarrow_{rm}^* x z$ in G , where $k:z = v$, there exists $\tilde{\pi} (= \pi \pi'')$ such that $S' \Rightarrow_{rm}^{\pi} \alpha z \Rightarrow_{rm}^{\pi''} \gamma_1 \cdots \gamma_{n-1} \delta_n y z \Rightarrow_{rm}^* x z$ in G . The derivation $\tilde{\pi}$ is always composed of $\tilde{\pi}_1$ and $\tilde{\pi}_2$ such that $\tilde{\pi}_1 = S' \Rightarrow_{rm}^* \gamma_1 \cdots \gamma_{n-1} B_{n-1} z_2$ and $\tilde{\pi}_2 = B_{n-1} w_{n-1} \Rightarrow_{B_{n-1}, W_{n-1}}^* \gamma_n z_1 w_{n-1} \Rightarrow_{B_{n-1}, W_{n-1}}^* \delta_n y z_1 w_{n-1}$, where $w_{n-1} = k:z_2$. At this point, if we set $\beta = \gamma_1 \cdots \gamma_{n-1}$, $B = B_n$, $W = W_n$, $\gamma = \gamma_n$, and $U = U_n$, then from Statement 2, we obtain the property that whenever there exists π such that $S' \Rightarrow_{rm}^{\pi} \alpha z \Rightarrow_{rm}^* x z$ in G , where $k:z = v$, there exist π^1 and π^2 , where $\pi^1 \pi^2 = \pi$ such

that $S' \Rightarrow_{rm}^{\pi^1} \beta Bz''$ and $Bw \Rightarrow_{B,W}^{\pi^2} \gamma z'w$ in G , where $w = k:z''$ and $z'z'' = z$. Here, $|\gamma|$ is bounded since it is a component of the nonterminal $[B, W, \gamma, U]$. As a result, it is a contradiction of the assumption that α and v have no predictable B, W , and γ satisfying Condition 1.

(If part) Assume that Statement 1 is true, but Algorithm 1 with G does not successfully terminate. It means that an indivisible cyclic nonterminal is generated, while Algorithm 1 with G works. Let $[A, R, \alpha, U]$ be such a nonterminal. Suppose that n is the constant in Statement 1. Without loss of generality, we can assume $|\alpha| > n$.

Take an arbitrary derivation

$$(6.6) \quad Ar \Rightarrow_{A,R}^* \alpha yr \Rightarrow_{A,R}^* \bar{x} yr$$

in G , where $r \in R$. Let $v = k:yr$. Then there exists $S' \Rightarrow_{rm}^* \theta Az \Rightarrow_{rm}^* \theta \alpha yz \Rightarrow_{rm}^* xyz$ in G , where $k:z = r$, $k:yz = v$, and $x = \hat{x}\bar{x}$. If we consider the viable prefix $\theta\alpha$, the string $x, x \in L^G(\theta\alpha)$, and the string $u, u \in RC_k^G(\theta\alpha)$, then there exist B, W , and γ , where $\theta\alpha = \beta\gamma$ and $|\gamma| \leq n$ satisfying Condition 1 since Statement 1 is true. That is, whenever there exists π such that $S' \Rightarrow_{rm}^{\pi} \beta\gamma z \Rightarrow_{rm}^* xyz$ in G , where $k:z = v$, there exist π^1 and π^2 , where $\pi^1\pi^2 = \pi$ such that $S' \Rightarrow_{rm}^{\pi^1} \beta Bz''$ and $Bw \Rightarrow_{B,W}^{\pi^2} \gamma z'w$ in G , where $z'z'' = z$ and $w = k:z''$. Hence, we have the property that: whenever there exists a derivation (6.6), there exist $Ar \Rightarrow_{A,R}^* \delta Bz'''r$ and $Bw \Rightarrow_{B,W}^* \gamma z'w$, where $z'z''' = y$ and $\alpha = \delta\gamma$. Consequently, we have the property: whenever there exists $Ar \Rightarrow_{A,R}^* \delta\gamma yr$, where $r \in R$ and $k:yr = v$, there exist $Ar \Rightarrow_{A,R}^* \delta Bz'''r$ and $Bw \Rightarrow_{B,W}^* \gamma z'w$, where $z'z''' = y$ and $w = k:z'''r$. In accordance with Lemma 4.4, $(A, R, \delta\gamma)\mathbb{I}_v(B, W', \gamma)$ holds for some W' , and hence, there exist B', W'' , and γ' such that $\Phi(A, R, \delta\gamma, v) = (B', W'', \gamma')$ holds. Then we have a contradiction to that $[A, R, \alpha, U]$ is indivisible cyclic. \square

In other words, the transformable grammars have a fixed constant n : when the stack string is $\alpha (\neq \epsilon)$, the input string already read is x , and the k -length prefix of the remaining input is v , we always have $B, W, \gamma (\neq \epsilon)$, where $|\gamma| \leq n$ such that γ and some prefix of the remaining input will be reduced to (B, W) . Figure 6.1 describes this situation; the left side is expected to be the right.

We define the transformable grammars using Algorithm 1 as extended PLR(k) grammars.

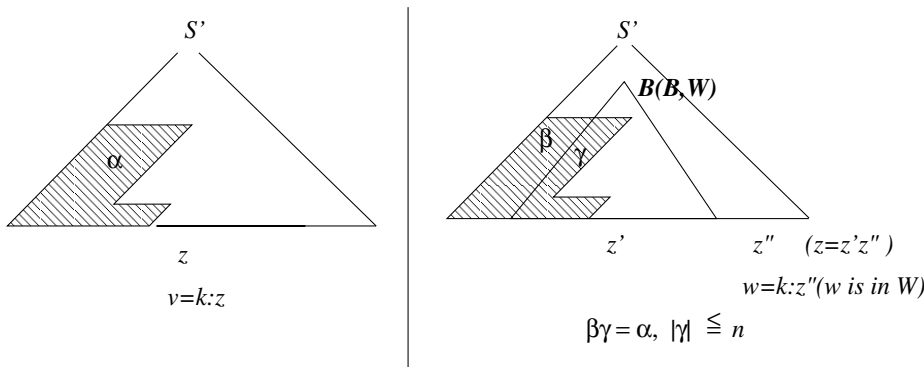


FIG. 6.1. Derivation trees.

DEFINITION 6.9. G is an extended $PLR(k)$ grammar iff Statement 1 is true.

We next relate extended $PLR(k)$ grammars with $PLR(k)$ grammars and k -transformable grammars.

LEMMA 6.10. Let G be $PLR(k)$. Then G is extended $PLR(k)$.

Proof. Suppose that $\alpha(\neq \epsilon)$ is a viable prefix of G , x is a string in $L^G(\alpha)$, and v is a string in $RC_k^G(\alpha)$. Then there exists a derivation $S' \Rightarrow_{rm}^* \beta Bz \Rightarrow_{rm} \beta X\delta\zeta z \Rightarrow_{rm}^* \beta X\delta y_2 z \Rightarrow_{rm}^* \beta X y_1 y_2 z \Rightarrow_{rm}^* x y_2 z$ in G , where $\beta X\delta = \alpha$, $k:y_2 z = v$, and $X\delta \neq \epsilon$. On the other hand, according to the definition of $PLR(k)$ grammars,

$$\text{if there exists } S' \Rightarrow_{rm}^* \beta' Az' \Rightarrow_{rm} \beta' \beta'' X\delta' \zeta' z' \Rightarrow_{rm}^* \beta' \beta'' X\delta' y_2' z'$$

$$\Rightarrow_{rm}^* \beta' \beta'' X y_1' y_2' z' \text{ in } G, \text{ where } \beta' \beta'' = \beta \text{ and } k:y_1' y_2' z' = k:y_1 y_2 z,$$

$$(6.7) \quad \text{then } A = B \text{ and } \beta'' = \epsilon.$$

Let n be the maximum length of the right-side string of a rule of G and γ be $X\delta$. Then we know that $|\gamma| \leq n(\gamma \neq \epsilon)$. On the other hand, whenever there exists $S' \Rightarrow_{rm}^* \beta\gamma z \Rightarrow_{rm}^* xz$ in G , there exists $S' \Rightarrow_{rm}^* \beta\gamma z \Rightarrow_{rm}^* \beta Xyz \Rightarrow_{rm}^* xz$ in G and, according to (6.7), there exist $S' \Rightarrow_{rm}^* \beta Bz_2$ and $Bw \Rightarrow_{rm}^* \gamma z_1 w$ in G , where $w = k:z_2$ and $z_1 z_2 = z$. Let us set W to be $FOLLOW_k(B)$. Then we have the property that whenever there exists π such that $S' \Rightarrow_{rm}^* \beta\gamma z \Rightarrow_{rm}^* xz$ in G , there exist π^1 and π^2 such that $S' \Rightarrow_{rm}^* \beta Bz_2$ and $Bw \Rightarrow_{B,W}^{\pi^2} \gamma z_1 w$ in G , where $\pi^1 \pi^2 = \pi$, $w = k:z_2$, and $z_1 z_2 = z$. In all, G is extended $PLR(k)$. \square

LEMMA 6.11. Let G be k -transformable. Then G is extended $PLR(k)$.

Proof. Since G is k -transformable, there exists a constant n such that if $\alpha(\neq \epsilon)$ is a viable prefix of G and v is a string in $RC_k^G(\alpha)$, then there exist B, W , and $\gamma(\neq \epsilon)$, where $\alpha = \beta\gamma$, $|\gamma| \leq n$, and $v \in RC_k^{B,W}(\gamma)$ such that whenever there exists $S' \Rightarrow_{rm}^* \beta\gamma z \Rightarrow_{rm}^* \beta yz$ in G , where $k:yz \in RC_k^{B,W}(\epsilon)$, there exist $S' \Rightarrow_{rm}^* \beta Bz''$ and $Bw \Rightarrow_{B,W}^* \gamma z' w \Rightarrow_{B,W}^* yz' w$ in G , where $w = k:z''(w \in W)$ and $z' z'' = z$. Note that the condition $k:yz \in RC_k^{B,W}(\epsilon)$ implies $k:z \in RC_k^{B,W}(\gamma)$. Hence, we have the property with the above n such that if $\alpha(\neq \epsilon)$ is a viable prefix of G , $x \in L^G(\alpha)$, and $v \in RC_k^G(\alpha)$, then there exist B, W , and $\gamma(\neq \epsilon)$, where $\alpha = \beta\gamma$, $|\gamma| \leq n$, and $v \in RC_k^{B,W}(\gamma)$ such that whenever there exists π such that $S' \Rightarrow_{rm}^* \beta\gamma z \Rightarrow_{rm}^* xz$ in G , where $k:z \in RC_k^{B,W}(\gamma)$, there exist π^1 and π^2 , where $\pi^1 \pi^2 = \pi$ such that $S' \Rightarrow_{rm}^* \beta Bz''$ and $Bw \Rightarrow_{B,W}^{\pi^2} \gamma z' w$ in G , where $w = k:z''(w \in W)$ and $z' z'' = z$. Thus, G is extended $PLR(k)$. \square

We showed that extended $PLR(k)$ grammars completely contain $PLR(k)$ grammars and k -transformable grammars. Extended $PLR(k)$ grammars, moreover, are larger than k -transformable grammars and $PLR(k)$ grammars. It can be proved by the following example.

Example 6.1. Let $G3 = (\{S, A\}, \{a, d, b, c\}, \{S \rightarrow aAd, S \rightarrow aB, B \rightarrow aA, A \rightarrow ab, A \rightarrow aA, A \rightarrow bc\}, S)$. Note that $G3$ is $LR(1)$ and extended $PLR(1)$. However, $G3$ is neither k -transformable nor $PLR(k)$ for all $k \geq 1$.

THEOREM 6.12. Extended $PLR(k)$ grammars are larger than $PLR(k)$ grammars and k -transformable grammars.

7. Concluding remarks. The contribution of this paper can be summarized as follows. First, we showed the incomparable relationship between $PLR(k)$ grammars

and k -transformable grammars. Second, we presented the generalization of the reduction goal prediction in LR parsing. We believe that the proposed one is the uppermost of the reduction goal prediction, which is performed by keeping the LR stack and investigating the k -length prefix of the remaining input. Extended $PLR(k)$ grammars are thus thought of as the uppermost class of $LL(k)$ covering transformable grammars based on such a prediction. Third, we showed that $LL(k)$ covering grammars can be deterministically constructed by defining the \mathbb{III} relation and investigating the orderable property of the relation. As a result, we can decide the transformableness of a given grammar in a single process. Lastly, we characterized the transformable grammars using grammatical derivation.

Acknowledgments. The authors deeply appreciate many helpful comments from an anonymous referee and coordinating efforts of the editor.

REFERENCES

- [1] A. V. AHO AND J. D. ULLMAN, *The Theory of Parsing, Translation, and Compiling*, Vols. 1 & 2, Prentice-Hall, Englewood Cliffs, NJ, 1972, 1973.
- [2] C. N. FISCHER AND R. J. LEBLANC, *Crafting a Compiler with C*, Benjamin/Cummings, Redwood City, CA, 1991.
- [3] M. HAMMER, *A New Grammatical Transformation into Deterministic Top Down Form*, Project MAC Technical report TR-119, MIT, Cambridge, MA, 1974.
- [4] M. HAMMER, *A new grammatical transformation into $LL(k)$ form*, in Proceedings of the Sixth Annual ACM Symposium on Theory of Computing, ACM, New York, 1974, pp. 266–275.
- [5] A. JOHNSTONE AND E. SCOTT, *Generalised recursive descent parsing and follow-determinism*, in Proceedings of the 7th International Conference on Compiler Construction, Lecture Notes in Comput. Sci. 1383, Springer-Verlag, Berlin, 1998, pp. 16–30.
- [6] D. E. KNUTH, *Top down syntax*, Acta Inform., 1 (1971), pp. 79–110.
- [7] G.-O. LEE AND K.-M. CHOE, *Characterization of $LL(k)$ Languages*, Technical report PSL-TR-2001-1, Department of Information Science and Telecommunications, Programming System Laboratory, Hanshin University, Republic of Korea, 2001.
- [8] T. J. PARR AND R. W. QUONG, *Adding semantic and syntactic predicates to $LL(k)$: Predicting $LL(k)$* , in Proceedings of the 5th International Conference on Compiler Construction, Lecture Notes in Comput. Sci. 786, Springer-Verlag, Berlin, 1994, pp. 263–277.
- [9] S. SIPPU AND E. SOISALON-SOININEN, *Parsing Theory*, Vols. I & II, Springer-Verlag, Berlin, Heidelberg, 1990.
- [10] E. SOISALON-SOININEN AND E. UKKONEN, *A method for transforming grammars into $LL(k)$ form*, Acta Inform., 12 (1979), pp. 338–369.
- [11] A. RUBMANN, *Dynamic $LL(k)$ parsing*, Acta Inform., 34 (1997), pp. 267–289.

COMPRESSED SUFFIX ARRAYS AND SUFFIX TREES WITH APPLICATIONS TO TEXT INDEXING AND STRING MATCHING*

ROBERTO GROSSI[†] AND JEFFREY SCOTT VITTER[‡]

Abstract. The proliferation of online text, such as found on the World Wide Web and in online databases, motivates the need for space-efficient text indexing methods that support fast string searching. We model this scenario as follows: Consider a text T consisting of n symbols drawn from a fixed alphabet Σ . The text T can be represented in $n \lg |\Sigma|$ bits by encoding each symbol with $\lg |\Sigma|$ bits. The goal is to support fast online queries for searching any string pattern P of m symbols, with T being fully scanned only once, namely, when the index is created at preprocessing time.

The text indexing schemes published in the literature are greedy in terms of space usage: they require $\Omega(n \lg n)$ additional bits of space in the worst case. For example, in the standard unit cost RAM, suffix trees and suffix arrays need $\Omega(n)$ memory words, each of $\Omega(\lg n)$ bits. These indexes are larger than the text itself by a multiplicative factor of $\Omega(\lg |\Sigma| n)$, which is significant when Σ is of constant size, such as in ASCII or UNICODE. On the other hand, these indexes support fast searching, either in $O(m \lg |\Sigma|)$ time or in $O(m + \lg n)$ time, plus an output-sensitive cost $O(occ)$ for listing the occ pattern occurrences.

We present a new text index that is based upon compressed representations of suffix arrays and suffix trees. It achieves a fast $O(m/\lg |\Sigma| n + \lg_{|\Sigma|}^{\epsilon} n)$ search time in the worst case, for any constant $0 < \epsilon \leq 1$, using at most $(\epsilon^{-1} + O(1)) n \lg |\Sigma|$ bits of storage. Our result thus presents for the first time an efficient index whose size is provably linear in the size of the text in the worst case, and for many scenarios, the space is actually sublinear in practice. As a concrete example, the compressed suffix array for a typical 100 MB ASCII file can require 30–40 MB or less, while the raw suffix array requires 500 MB. Our theoretical bounds improve *both* time and space of previous indexing schemes. Listing the pattern occurrences introduces a sublogarithmic slowdown factor in the output-sensitive cost, giving $O(occ \lg_{|\Sigma|}^{\epsilon} n)$ time as a result. When the patterns are sufficiently long, we can use auxiliary data structures in $O(n \lg |\Sigma|)$ bits to obtain a total search bound of $O(m/\lg |\Sigma| n + occ)$ time, which is optimal.

Key words. compression, text indexing, text retrieval, compressed data structures, suffix arrays, suffix trees, string searching, pattern matching

AMS subject classifications. 68W05, 68Q25, 68P05, 68P10, 68P30

DOI. 10.1137/S0097539702402354

1. Introduction. A great deal of textual information is available in electronic form in online databases and on the World Wide Web, and therefore devising efficient text indexing methods to support fast string searching is an important topic for investigation. A typical search scenario involves string matching in a text string T of length n [49]: given an input pattern string P of length m , the goal is to find occurrences of P in T . Each symbol in P and T belongs to a fixed alphabet Σ of size $|\Sigma| \leq n$. An occurrence of the pattern at position i means that the substring $T[i, i + m - 1]$ is equal to P , where $T[i, j]$ denotes the concatenation of the symbols

*Received by the editors February 9, 2002; accepted for publication (in revised form) May 2, 2005; published electronically October 17, 2005. A preliminary version of these results appears in [37].

<http://www.siam.org/journals/sicomp/35-2/40235.html>

[†]Dipartimento di Informatica, Università di Pisa, Largo B. Pontecorvo 3, 56127 Pisa, Italy (grossi@di.unipi.it). This author's work was supported in part by the United Nations Educational, Scientific and Cultural Organization (UNESCO) and by the Italian Ministry of Research and Education (MIUR).

[‡]Department of Computer Sciences, Purdue University, West Lafayette, IN 47907–2066 (jsv@purdue.edu). Part of this work was done while the author was at Duke University and on sabbatical at INRIA in Sophia Antipolis, France. It was supported in part by Army Research Office MURI grants DAAH04–96–1–0013, DAAD19–01–1–0725, and DAAD19–03–1–0321 and by National Science Foundation research grants CCR–9877133 and IIS–0415097.

in T at positions $i, i + 1, \dots, j$.

In this paper, we consider three types of string matching queries: existential, counting, and enumerative. An existential query returns a Boolean value that indicates whether P is contained in T . A counting query computes the number occ of occurrences of P in T , where $occ \leq n$. An enumerative query outputs the list of occ positions, where P occurs in T . Efficient offline string matching algorithms, such as that of Knuth, Morris, and Pratt [49], can answer each individual query in $O(m + n)$ time via an efficient text scan.

The large mass of existing online text documents makes it infeasible to scan through all the documents for every query, because n is typically much larger than the pattern length m and the number of occurrences occ . In this scenario, text indexes are preferable, as they are especially efficient when several string searches are to be performed on the same set of text documents. The text T needs to be entirely scanned only once, namely, at preprocessing time when the indexes are created. After that, searching is output-sensitive, that is, the time complexity of each online query is proportional to either $O(m \lg |\Sigma| + occ)$ or $O(m + \lg n + occ)$, which is much less than $\Theta(m + n)$ when n is sufficiently large.

The most popular indexes currently in use are inverted lists and signature files [48]. Inverted lists are theoretically and practically superior to signature files [72]. Their versatility allows for several kinds of queries (exact, Boolean, ranked, and so on) whose answers have a variety of output formats. They are efficient indexes for texts that are structured as long sequences of terms (or words) in which T is partitioned into nonoverlapping substrings $T[i_k, j_k]$ (the terms), where $1 \leq i_k \leq j_k < i_{k+1} \leq n$. We refer to the set of terms as the vocabulary. For each distinct term in the vocabulary, the index maintains the inverted list (or position list) $\{i_k\}$ of the occurrences of that term in T . As a result, in order to search efficiently, search queries must be limited to terms or prefixes of them; it does not allow for efficient searching of arbitrary substrings of the text as in the string matching problem. For this reason, inverted files are sometimes referred to as *term-level* or *word-level* text indexes.

Searching *unstructured* text to answer string matching queries adds a new difficulty to text indexing. This case arises with DNA sequences and in some Eastern languages (Burmese, Chinese, Taiwanese, Tibetan, etc.), which do not have a well-defined notion of terms. The set of successful search keys is possibly much larger than the set of terms in structured texts, because it consists of all feasible substrings of T ; that is, we can have as many as $\binom{n}{2} = \Theta(n^2)$ distinct substrings in the worst case, while the number of distinct terms is at most n (considered as nonoverlapping substrings). Suffix arrays [55, 35], suffix trees [57, 68], and similar tries or automata [20] are among the prominent data structures used for unstructured texts. Since they can handle all the search keys in $O(n)$ memory words, they are sometimes referred to as *full-text* indexes.

The *suffix tree* for text $T = T[1, n]$ is a compact trie whose n leaves represent the n text suffixes $T[1, n], T[2, n], \dots, T[n, n]$. By “compact” we mean that each internal node has at least two children. Each edge in the tree is labeled with one or more symbols for purposes of search navigation. The leaf with value ℓ represents the suffix $T[\ell, n]$. The leaf values in an in-order traversal of the tree represent the n suffixes of T in lexicographic order. An example suffix tree appears in Figure 1.

A *suffix array* $SA = SA[1, n]$ for text $T = T[1, n]$ consists of the values of the leaves of the suffix tree in in-order, but without the tree structure information. In other words, $SA[i] = \ell$ means that $T[\ell, n]$ is the i th smallest suffix of T in lexicographic order. The suffix array corresponding to the suffix tree of Figure 1 appears in Figure 2.

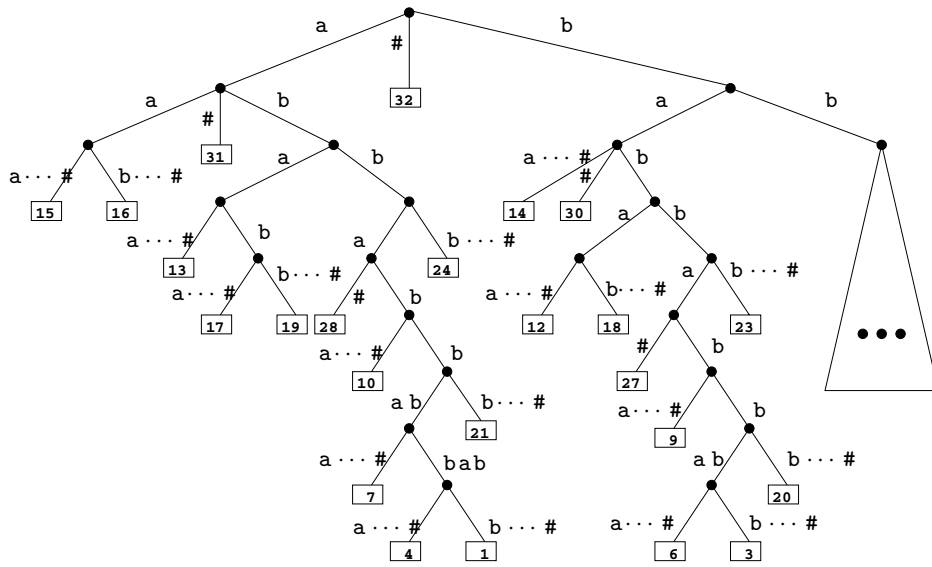


FIG. 1. Suffix tree built on text $T = \text{abbabbabbabbabaabababbabbabba}\#$ of length $n = 32$, where the last character is an end-of-string symbol $\#$. The rightmost subtree (the triangle representing the suffixes of the form $bb\cdots\#$) is not expanded in the figure. The edge label $a\cdots\#$ or $b\cdots\#$ on the edge leading to the leaf with value l denotes the remaining characters of the suffix $T[l, n]$ that have not already been traversed. For example, the first suffix in lexicographic format is the suffix $T[15, n]$, namely, $aaabababbabbabba\#$, and the last edge represents the 16-symbol substring that follows the prefix aa .

To speed up searches, a separate array is often maintained, which contains auxiliary information such as the lengths of the longest common prefixes of a subset of the suffixes [55].

Suffix trees and suffix arrays organize the suffixes so as to support the efficient search of their prefixes. Given a search pattern P , in order to find an occurrence $T[i, i + m - 1] = P$, we can exploit the property that P must be the prefix of suffix $T[i, n]$. In general, existential and counting queries take $O(m \lg |\Sigma|)$ time using automata or suffix trees and their variations, and they take $O(m + \lg n)$ time using suffix arrays along with longest common prefixes. Enumerative queries take an additive output-sensitive cost $O(occ)$. In this paper, we use the term “suffix array” to denote the array containing the permutation of positions $1, 2, \dots, n$, but without the longest common prefix information mentioned above. Full-text indexes such as suffix arrays are more powerful than term-level inverted lists, since full-text indexes can also implement inverted lists efficiently by storing the suffixes $T[i_k, n]$ that correspond to the occurrences of the terms.

1.1. Issues on space efficiency. Suffix arrays and suffix trees are data structures with increasing importance because of the growing list of their applications. Besides string searching, they also have significant use in molecular biology, data compression, data mining, and text retrieval, to name but a few applications [7, 38, 55]. However, the sizes of the data sets in these applications can become extremely large, and space occupancy is often a critical issue. A major disadvantage that limits the applicability of text indexes based upon suffix arrays and suffix trees is that they occupy significantly more space than do inverted lists.

1	15	aaabababbabbabba#
2	16	aabababbabbabbabba#
3	31	a#
4	13	abaaabababbabbabba#
5	17	abababbabbabbabba#
6	19	ababbabbabbabba#
7	28	abba#
8	10	abbabaaabababbabbabba#
9	7	abbabbabaaabababbabbabba#
10	4	abbabbabbabaaabababbabbabba#
11	1	abbabbabbabbabaaabababbabbabba#
12	21	abbabbabbabba#
13	24	abbbabba#
14	32	#
15	14	baaabababbabbabba#
16	30	ba#
17	12	babaaabababbabbabba#
18	18	bababbabbabbabba#
19	27	babba#
20	9	babbabaaabababbabbabba#
21	6	babbabbabaaabababbabbabba#
22	3	babbabbabbabaaabababbabbabba#
23	20	babbabbabbabba#
24	23	babbabbabba#
⋮	⋮	⋮
32	25	bbbabba#

FIG. 2. Suffix array for the text T shown in Figure 1, where $a < \# < b$. Note that the array values correspond to the leaf values in the suffix tree in Figure 1 traversed in in-order.

We can illustrate this point by a more careful accounting of the space requirements in the unit cost RAM model. We assume that each symbol in the text T is encoded by $\lg |\Sigma|$ bits, for a total of $n \lg |\Sigma|$ bits.¹ In suffix arrays, the positions of the n suffixes of T are stored as a permutation of $1, 2, \dots, n$, using $n \lg n$ bits (kept in an array consisting of n words, each of $\lg n$ bits). Suffix trees require considerably more space: between $4n \lg n$ and $5n \lg n$ bits (stored in $4n-5n$ words) [55]. In contrast, inverted lists require only approximately 10% of the text size [58], and thus suffix arrays and suffix trees require significantly more bits. From a theoretical point of view, if the alphabet is very large, namely, if $\lg |\Sigma| = \Theta(\lg n)$, then suffix arrays require roughly the same number of bits as the text. However, in practice, the alphabet size $|\Sigma|$ is typically a fixed constant, such as $|\Sigma| = 256$ in electronic documents in ASCII or larger in UNICODE format, and $\Sigma = 4$ in DNA sequences. In such cases in practice, suffix arrays and suffix trees are larger than the text by a significant multiplicative factor of $\Theta(\lg_{|\Sigma|} n) = \Theta(\lg n)$. For example, a DNA sequence of n symbols (with $|\Sigma| = 4$) can be stored with $2n$ bits in a computer. The suffix array for the sequence requires instead at least n words of 4 bytes each, or $32n$ bits, which is 16 times larger than the text itself. On the other hand, we cannot resort to inverted files since they do not support a general search on unstructured sequences.

¹In this paper, we use the notation $\lg_b^c n = (\log_b n)^c = (\log n / \log b)^c$ to denote the c th power of the base- b logarithm of n . If no base b is specified, the implied base is 2.

In [62], Munro, Raman, and Rao solve the open question raised by Muthukrishnan by showing how to represent suffix trees in $n \lg n + O(n)$ bits while allowing $O(m)$ -time search of binary pattern strings of length m . This result highlights the conceptual barrier of $n \lg n$ bits of storage needed for text indexing. In this paper, we go one step further and investigate whether it is possible to design a full-text index in $o(n \lg n)$ bits, while still supporting efficient search.

The question of space usage is important in both theory and practice. Prior to our work, the state of the art has taken for granted that at least $n \lg n$ bits are needed to represent the permutation of the text positions for any efficient full-text index. On the other hand, if we note that each text of n symbols is in one-to-one correspondence with a suffix array, then we can easily see by a simple information-theoretic argument that $\Omega(n \lg |\Sigma|)$ bits are required to represent the permutation. The argument is based upon the fact that there are $|\Sigma|^n$ different text strings of length n over the alphabet Σ ; hence, there are that many different suffix arrays, and we need $\Omega(n \lg |\Sigma|)$ bits to distinguish them from one another. It is therefore an interesting problem to close this gap in order to see if there is an efficient representation of suffix arrays that use nearly $n \lg |\Sigma| + O(n)$ bits in the worst case, even for random strings.

In order to have an idea of the computational difficulty of the question, let us follow a simple approach that saves space. Let us consider binary alphabets. We bunch every $\lg n$ bits together into a word (in effect, constructing a large alphabet) and create a text of length $n/\lg n$ and a pattern of length $m/\lg n$. The suffix array on the new text requires $O((n/\lg n) \lg n) = O(n)$ bits. Searching for a pattern of length m must also consider situations when the pattern is not aligned at the precise word boundaries. What is the searching cost? It appears that we have to handle $\lg n$ situations, with a slowdown factor of $\lg n$ in the time complexity of the search. However, this is not really so; we actually have to pay a much larger slowdown factor of $\Omega(n)$ in the search cost, which makes querying the text index more expensive than running the $O(m+n)$ -time algorithms from scratch, such as in [49]. To see why, let us examine the situation in which the pattern occurs k positions to the right of a word boundary in the text. In order to query the index, we have to align the pattern with the boundary by padding k bits to the left of the pattern. Since we do not know the correct k bits to prepend to the pattern, we must try all 2^k possible settings of the k bits. When $k \approx \lg n$, we have to query the index $2^k = \Omega(n)$ times in the worst case. (See the sparse suffix trees [47] cited in section 1.3 to partially alleviate this drawback.)

The above example shows that a small reduction in the index size can make querying the index useless in the worst case, as it can cost at least as much as performing a full scan of the text from scratch. In section 1.3, we describe previous results motivated by the need to find an efficient solution to the problem of designing a full-text index that saves space and time in the worst case. No data structures with the functionality of suffix trees and suffix arrays that have appeared in the literature to date use $\Theta(n \lg |\Sigma|) + o(n \lg n)$ bits and support fast queries in $o(m \lg |\Sigma|)$ or $o(m + \lg n)$ worst-case time. Our goal in this paper is to simultaneously reduce *both* the space bound *and* the query time bound.

1.2. Our results. In this paper, we begin the study of the compressibility of suffix arrays and related full-text indexes. We assume for simplicity that the alphabet Σ is of bounded size (i.e., ASCII or UNICODE/UTF8). We recall that the suffix array SA for text T stores the suffixes of T in lexicographic order, as shown in Figure 2. We represent SA in the form of a permutation of the starting positions, $1, 2, \dots, n$, of the

suffixes in T . For all $1 \leq i < j \leq n$, we have $T[SA[i], n] < T[SA[j], n]$ in lexicographic order. We call each entry in SA a *suffix pointer*.

Given a text T and its suffix array SA , we consider the problem of obtaining a *compressed suffix array* from both T and SA so as to support the following two basic operations:

1. *compress*(T, SA): Compress SA to obtain its succinct representation. After that, text T is retained while SA can be discarded.
2. *lookup*(i): Given the compressed representation mentioned above, return $SA[i]$, which is the suffix pointer in T of the i th suffix $T[SA[i], n]$ in lexicographic order.

(More functionalities are introduced in [64].) The primary measures of performance are the query time to do *lookup*, the amount of space occupied by the compressed suffix array, and the preprocessing time and space taken by *compress*.

In this paper, we exploit the “implicit structure” underlying the permutation of the suffix pointers stored in SA , which takes advantage of the fact that not all permutations are valid suffix arrays. For any fixed value of $0 < \epsilon \leq 1$, we show how to implement operation *compress* in $(1 + \epsilon^{-1}) n \lg |\Sigma| + o(n \lg |\Sigma|)$ bits so that each call to *lookup* takes sublogarithmic worst-case time, that is, $O(\lg_{|\Sigma|}^{\epsilon} n)$ time. We can also achieve $(1 + \frac{1}{2} \lg \lg_{|\Sigma|} n) n \lg |\Sigma| + O(n)$ bits so that calls to *lookup* can be done in $O(\lg \lg_{|\Sigma|} n)$ time. The preprocessing time is $O(n \lg |\Sigma|)$. Note that the auxiliary space during preprocessing is larger, i.e., $O(n \lg n)$ bits, since our preprocessing requires the suffix array in uncompressed form to output it in compressed form. Our findings have several implications as follows:

1. When $|\Sigma| = O(2^{o(\lg n)})$, we break the space barrier of $\Omega(n \lg n)$ bits for a suffix array while retaining $o(\lg n)$ lookup time in the worst case. We refer the reader to the literature described in section 1.3.
2. We can implement a form of compressed suffix trees in $2n \lg |\Sigma| + O(n)$ bits by using compressed suffix arrays (with $\epsilon = 1$) and the techniques for compact representation of Patricia tries presented in [62]. They occupy asymptotically up to a small constant factor the *same* space as that of the text string being indexed.
3. Our compressed suffix arrays and compressed suffix trees are provably as good as inverted lists in terms of space usage, at least theoretically. In the worst case, they require asymptotically the *same* number of bits.
4. We can build a hybrid full-text index on T in at most $(\epsilon^{-1} + O(1)) n \lg |\Sigma|$ bits by a suitable combination of our compressed suffix trees and previous techniques [17, 45, 62, 59]. We can answer existential and counting queries of any pattern string of length m in $O(m/\lg_{|\Sigma|} n + \lg_{|\Sigma|}^{\epsilon} n)$ search time in the worst case, which is $o(\min\{m \lg |\Sigma|, m + \lg n\})$, smaller than previous search bounds. For enumerative queries, we introduce a sublogarithmic slowdown factor in the output-sensitive cost, giving $O(occ \lg_{|\Sigma|}^{\epsilon} n)$ time as a result. When the patterns are sufficiently long, namely, for $m = \Omega((\lg_{|\Sigma|}^{1+\epsilon} n)(\lg_{|\Sigma|} \lg n))$, we can use auxiliary data structures in $O(n \lg |\Sigma|)$ bits to obtain a total search bound of $O(m/\lg_{|\Sigma|} n + occ)$ time, which is optimal.

The bounds claimed in point 4 need further elaboration. Specifically, searching takes $O(1)$ time for $m = o(\lg n)$, and $O(m/\lg_{|\Sigma|} n + \lg_{|\Sigma|}^{\epsilon} n) = o(m \lg |\Sigma|)$ time otherwise. That is, we achieve optimal $O(m/\lg_{|\Sigma|} n)$ search time for sufficiently large $m = \Omega(\lg_{|\Sigma|}^{1+\epsilon} n)$. For enumerative queries, retrieving all occ occurrences has cost $O(m/\lg_{|\Sigma|} n + occ \lg_{|\Sigma|}^{\epsilon} n)$ when both conditions $m \in [\epsilon \lg n, o(\lg^{1+\epsilon} n)]$ and $occ = o(n^{\epsilon})$ hold, and cost $O(m/\lg_{|\Sigma|} n + occ + (\lg^{1+\epsilon} n)(\lg |\Sigma| + \lg \lg n))$ otherwise.

The results described in this paper are theoretical, but they also have substantial practical value. The ideas described here, with the extensions described by Sadakane [64], have been tested experimentally in further work discussed in section 1.3. Central to our algorithms is the definition of function Φ and its implications described in section 2. This function is at the heart of many subsequent papers on compressed text indexing (such as suffix links in suffix trees, binary search trees in sorted dictionaries, and join operators in relational databases), thus is the major by-product of the findings presented in this paper. Ultimately Φ is related to sorting-based compression because it represents the inverse of the last-to-first mapping for the Burrows–Wheeler transform [14]. We refer the interested reader to section 1.3 for the state of the art in compressed text indexing and to section 2 for a discussion of the function Φ .

1.3. Related work. The seminal paper by Knuth, Morris, and Pratt [49] provides the first string matching solution taking $O(m+n)$ time and $O(m)$ words to scan the text. The space requirement was remarkably lowered to $O(1)$ words in [33, 19]. The new paradigm of compressed pattern matching was introduced in [2] and explored for efficiently scanning compressed texts in [3, 24]. When many queries are to be performed on the same text, it is better to resort to text indexing. A relevant paper [68] introduced a variant of the suffix tree for solving the text indexing problem in string matching. This paper pointed out the importance of text indexing as a tool to avoid a full scan of the text at each pattern search. This method takes $O(m \lg |\Sigma|)$ search time plus the output-sensitive cost $O(occ)$ to report the occurrences, where $occ \leq n$. Since then, a plethora of papers have studied the text indexing problem in several contexts, sometimes using different terminology [10, 11, 18, 28, 50, 41, 57, 55, 67]; for more references see [7, 20, 38]. Although very efficient, the resulting index data structures are greedy in terms of space, using at least n words or $\Omega(n \lg n)$ bits.

Numerous papers faced the problem of saving space in these data structures, both in practice and in theory. Many of the papers were aimed at improving the lower-order terms, as well as the constants in the higher-order terms, or at achieving tradeoff between space requirements and search time complexity. Some authors improved the multiplicative constants in the $O(n \lg n)$ -bit practical implementations. For the analysis of constants, we refer the reader to [6, 15, 34, 44, 53, 54, 55]. Other authors devised several variations of sparse suffix trees to store a subset of the suffixes [5, 35, 47, 46, 56, 59]. Some of them wanted queries to be efficient when the occurrences are aligned with the boundaries of the indexed suffixes. Sparsity saves much space but makes the search for arbitrary substrings difficult and, in the worst case, it is as expensive as scanning the whole text in $O(m+n)$ time. Another interesting index, the Lempel-Ziv index [45], occupies $O(n)$ bits and takes $O(m)$ time to search patterns shorter than $\lg n$ with an output-sensitive cost for reporting the occurrences; for longer patterns, it may occupy $\Omega(n \lg n)$ bits. An efficient and practical compressed index is discussed in [21], but its searches are at word level and are not full text (i.e., with arbitrary substrings).

An alternative line of research has been built upon succinct representation of trees in $2n$ bits, with navigational operations [42]. That representation was extended in [16] to represent a suffix tree in $n \lg n$ bits plus an extra $O(n \lg \lg n)$ expected number of bits. A solution requiring $n \lg n + O(n)$ bits and $O(m + \lg \lg n)$ search time was described in [17]. Munro, Raman, and Rao [62] used it along with an improved succinct representation of balanced parentheses [61] in order to get $O(m \lg |\Sigma|)$ search time with only $n \lg n + o(n)$ bits. They also show in [62] how to get $O(m)$ time and

$O(n \lg n / \lg \lg n)$ bits for existential queries in binary patterns.

The preliminary version of our results presented in [37] stimulated much further work; according to a search on <http://www.scholar.google.com>, as of May 2005 more than 80 interesting results have appeared citing this work. A first question raised concerns lower bounds. Assuming that the text is read-only and using a stronger version of the bit-probe model, Demaine and López-Ortiz [22] have shown in the worst case that any text index with alphabet size $|\Sigma| = 2$ that supports fast queries by probing $O(m)$ bits in the text must use $\Omega(n)$ bits of extra storage space. (See also Gál and Miltersen [32] for a general class of lower bounds.) Thus, our index is space-optimal in this sense. A second concerns compressible text. Ferragina and Manzini [29, 30] have devised the Fast Minute index (FM-index), based upon the Burrows–Wheeler transform [14], that asymptotically achieves the order- k empirical entropy of the text and allows them to obtain self-indexing texts (i.e., the compressed text and its index are the same sequence of bits). Sadakane [64] has shown that compressed suffix arrays can be used for self-indexing texts, with space bound by the order-0 entropy. (He also uses our Lemma 2 in section 3.1 to show how to store the skip values of the suffix tree in $O(n)$ bits [65].) The space of compressed suffix arrays has been further reduced to the order- k entropy (with a multiplicative constant of 1) by Grossi, Gupta, and Vitter [36] using a novel analysis based on a finite set model. Both the compressed suffix array and the FM-index require $O(n \lg n)$ auxiliary bits of space during preprocessing, so a third question arises concerning a space-efficient construction. Hon, Sadakane, and Sung [40] have shown how to build both the compressed suffix array and the FM-index with $O(n \lg |\Sigma|)$ bits of auxiliary space by using the text alone and small bookkeeping data structures. Numerous other papers have appeared as well, representing a recent new trend in text indexing, causing space efficiency to no longer be a major obstacle to the large-scale application of index data structures [71]. Ideally we’d like to find an index that uses as few as bits as possible and supports enumerative queries for each query pattern in sublinear time in the worst case (in addition to the output-sensitive cost).

1.4. Outline of the paper. In section 2 we describe the ideas behind our new data structure for compressed suffix arrays, including function Φ . Details of our compressed suffix array construction are given in section 3. In section 4 we show how to use compressed suffix arrays to construct compressed suffix trees and a general space-efficient indexing mechanism to speed up text search. We give final comments in section 5. We adopt the standard unit cost RAM for the analysis of our algorithms, as does the previous work with which we compare. We use standard arithmetic and Boolean operations on words of $O(\lg n)$ bits. Each operation takes constant time; each word is read or written in constant time.

2. Compressed suffix arrays. The compression of suffix arrays falls into the general framework presented by Jacobson [43] for the abstract optimization of data structures. We start from the specification of our data structure as an abstract data type with its supported operations. We take the time complexity of the “natural” (and less space efficient) implementation of the data structure. Then we define the class C_n of all distinct data structures storing n elements. A simple information-theoretic argument implies that each such data structure can be canonically identified by $\lg |C_n|$ bits. We try to give a succinct implementation of the same data structure in $O(\lg |C_n|)$ bits, while supporting the operations within time complexity comparable with that of the natural implementation. However, the information-theoretic argument alone does not guarantee that the operations can be supported efficiently.

We define the suffix array SA for a binary string T as an abstract data type that supports the two operations *compress* and *lookup* described in the introduction. We will adopt the convention that T is a binary string of length $n - 1$ over the alphabet $\Sigma = \{\mathbf{a}, \mathbf{b}\}$, and it is terminated in the n th position by a special end-of-string symbol $\#$, such that $\mathbf{a} < \# < \mathbf{b}$.² We will discuss the case of alphabets of size $|\Sigma| > 2$ at the end of the section.

The suffix array SA is a permutation of $\{1, 2, \dots, n\}$ that corresponds to the lexicographic ordering of the suffixes in T ; that is, $SA[i]$ is the starting position in T of the i th suffix in lexicographic order. The example below shows the suffix arrays corresponding to the 16 binary strings of length 4:

aaaa#	aaab#	aaba#	aabb#	abaa#	abab#	abba#	abbb#
12345	12354	14253	12543	34152	13524	41532	15432
baaa#	baab#	baba#	babb#	bbaa#	bbab#	bbba#	bbbb#
23451	23514	42531	25143	34521	35241	45321	54321

The natural explicit implementation of suffix arrays requires $O(n \lg n)$ bits and supports the *lookup* operation in constant time. The abstract optimization discussed above suggests that there is a canonical way to represent suffix arrays in $O(n)$ bits. This observation follows from the fact that the class C_n of suffix arrays has no more than 2^{n-1} distinct members, as there are 2^{n-1} binary strings of length $n - 1$. That is, not all the $n!$ permutations are necessarily suffix arrays.

We use the intuitive correspondence between suffix arrays of length n and binary strings of length $n - 1$. According to the correspondence, given a suffix array SA , we can infer its associated binary string T and vice versa. To see how, let x be the entry in SA corresponding to the last suffix $\#$ in lexicographic order. Then T must have the symbol \mathbf{a} in each of the positions pointed to by $SA[1], SA[2], \dots, SA[x - 1]$, and it must have the symbol \mathbf{b} in each of the positions pointed to by $SA[x + 1], SA[x + 2], \dots, SA[n]$. For example, in the suffix array $\langle 45321 \rangle$ (the 15th of the 16 examples above), the suffix $\#$ corresponds to the second entry 5. The preceding entry is 4, and thus the string T has \mathbf{a} in position 4. The subsequent entries are 3, 2, 1, and thus T must have \mathbf{b} s in positions 3, 2, 1. The resulting string T , therefore, must be **bbba#**.

The abstract optimization does not say anything regarding the efficiency of the supported operations. By the correspondence above, we can define a trivial *compress* operation that transforms SA into a sequence of $n - 1$ bits plus $\#$, namely, string T itself. The drawback, however, is the unaffordable cost of *lookup*. It takes $\Omega(n)$ time to decompress a single suffix pointer in SA , as it must build the whole suffix array on T from scratch. In other words, the trivial method proposed so far does not support efficient *lookup* operations.

In this section we describe an efficient method to represent suffix arrays in $O(n)$ bits with fast *lookup* operations. Our idea is to distinguish among the permutations of $\{1, 2, \dots, n\}$ by relating them to the suffixes of the corresponding strings, instead of studying them alone. We mimic a simple divide-and-conquer “deconstruction” of the suffix arrays to define the permutation for an arbitrary (e.g., random) string T recursively in terms of shorter permutations. For some examples of divide-and-conquer

²Usually, an end-of-symbol character is not explicitly stored in T , but rather is implicitly represented by a blank symbol \square , with the ordering $\square < \mathbf{a} < \mathbf{b}$. However, our use of $\#$ is convenient for showing the explicit correspondence between suffix arrays and binary strings.

construction of suffix arrays and suffix trees, see [8, 25, 26, 27, 55, 66]. We reverse the construction process to discover a recursive structure of the permutations that makes their compression possible. We describe the decomposition scheme in section 2.1, giving some intuition on the compression in section 2.2. We summarize the results thus obtained in section 2.3.

2.1. Decomposition scheme. Our decomposition scheme is by a simple recursion mechanism. Let SA be the suffix array for binary string T . In the base case, we denote SA by SA_0 , and let $n_0 = n$ be the number of its entries. For simplicity in exposition, we assume that n is a power of 2.

In the inductive phase $k \geq 0$, we start with suffix array SA_k , which is available by induction. It has $n_k = n/2^k$ entries and stores a permutation of $\{1, 2, \dots, n_k\}$. (Intuitively, this permutation is that resulting from sorting the suffixes of T whose suffix pointers are multiples of 2^k .) We run four main steps as follows to transform SA_k into an equivalent but more succinct representation:

Step 1. Produce a bit vector B_k of n_k bits such that $B_k[i] = \mathbf{1}$ if $SA_k[i]$ is even and $B_k[i] = \mathbf{0}$ if $SA_k[i]$ is odd.

Step 2. Map each $\mathbf{0}$ in B_k onto its companion $\mathbf{1}$. (We say that a certain $\mathbf{0}$ is the *companion* of a certain $\mathbf{1}$ if the odd entry in SA associated with the $\mathbf{0}$ is 1 less than the even entry in SA associated with the $\mathbf{1}$.) We can denote this correspondence by a partial function Ψ_k , where $\Psi_k(i) = j$ if and only if $SA_k[i]$ is odd and $SA_k[j] = SA_k[i] + 1$. When defined, $\Psi_k(i) = j$ implies that $B_k[i] = \mathbf{0}$ and $B_k[j] = \mathbf{1}$. It is convenient to make Ψ_k a total function by setting $\Psi_k(i) = i$ when $SA_k[i]$ is even (i.e., when $B_k[i] = \mathbf{1}$). In summary, for $1 \leq i \leq n_k$, we have

$$\Psi_k(i) = \begin{cases} j & \text{if } SA_k[i] \text{ is odd and } SA_k[j] = SA_k[i] + 1; \\ i & \text{otherwise.} \end{cases}$$

Step 3. Compute the number of 1's for each prefix of B_k . We use function $rank_k$ for this purpose; that is, $rank_k(j)$ counts how many 1's are in the first j bits of B_k .

Step 4. Pack together the even values from SA_k and divide each of them by 2. The resulting values form a permutation of $\{1, 2, \dots, n_{k+1}\}$, where $n_{k+1} = n_k/2 = n/2^{k+1}$. Store them in a new suffix array SA_{k+1} of n_{k+1} entries and remove the old suffix array SA_k .

The following example illustrates the effect of a single application of Steps 1–4. Here, $\Psi_0(25) = 16$ as $SA_0[25] = 29$ and $SA_0[16] = 30$. The new suffix array SA_1 explicitly stores the suffix pointers (divided by 2) for the suffixes that start at even positions in the original text T . For example, $SA_1[3] = 5$ means that the third lexicographically smallest suffix that starts at an even position in T is the one starting at position $2 \times 5 = 10$, namely, $abbabaa \dots \#$.

```

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32
T: a b b a b b a b b a b b a b a a b a b a b b a b b a b b a #
SA0: 15 16 31 13 17 19 28 10 7 4 1 21 24 32 14 30 12 18 27 9 6 3 20 23 29 11 26 8 5 2 22 25
B0: 0 1 0 0 0 0 1 1 0 1 0 0 1 1 1 1 1 0 0 1 0 1 0 0 0 1 1 0 1 1 0
rank0: 0 1 1 1 1 1 2 3 3 4 4 4 5 6 7 8 9 10 10 10 11 11 12 12 12 13 14 14 15 16 16
Psi0: 2 2 14 15 18 23 7 8 28 10 30 31 13 14 15 16 17 18 7 8 21 10 23 13 16 17 27 28 21 30 31 27
      1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
SA1: 8 14 5 2 12 16 7 15 6 9 3 10 13 4 1 11
    
```

```

procedure rlookup( $i, k$ ):
  if  $k = \ell$  then
    return  $SA_\ell[i]$ 
  else
    return  $2 \times \textit{rlookup}(\textit{rank}_k(\Psi_k(i)), k + 1) + (B_k[i] - 1)$ .

```

FIG. 3. Recursive lookup of entry $SA_k[i]$ in a compressed suffix array.

The next lemma shows that these steps preserve the information originally kept in suffix array SA_k .

LEMMA 1. *Given suffix array SA_k , let B_k , Ψ_k , \textit{rank}_k , and SA_{k+1} be the result of the transformation performed by Steps 1–4 of phase k . We can reconstruct SA_k from SA_{k+1} by the following formula for $1 \leq i \leq n_k$:*

$$SA_k[i] = 2 \cdot SA_{k+1}[\textit{rank}_k(\Psi_k(i))] + (B_k[i] - 1).$$

Proof. Suppose $B_k[i] = \mathbf{1}$. By Step 3, there are $\textit{rank}_k(i)$ 1's among $B_k[1], B_k[2], \dots, B_k[i]$. By Step 1, $SA_k[i]$ is even, and by Step 4, $SA_k[i]/2$ is stored in the $\textit{rank}_k(i)$ th entry of SA_{k+1} . In other words, $SA_k[i] = 2 \cdot SA_{k+1}[\textit{rank}_k(i)]$. As $\Psi_k(i) = i$ by Step 2, and $B_k[i] - 1 = 0$, we obtain the claimed formula.

Next, suppose that $B_k[i] = \mathbf{0}$ and let $j = \Psi_k(i)$. By Step 2, we have $SA_k[i] = SA_k[j] - 1$ and $B_k[j] = \mathbf{1}$. Consequently, we can apply the previous case of our analysis to index j , and we get $SA_k[j] = 2 \cdot SA_{k+1}[\textit{rank}_k(j)]$. The claimed formula follows by replacing j with $\Psi_k(i)$ and by noting that $B_k[i] - 1 = -1$. \square

In the previous example, $SA_0[25] = 2 \cdot SA_1[\textit{rank}_0(16)] - 1 = 2 \cdot 15 - 1 = 29$. We now give the main ideas to perform the compression of suffix array SA and support the lookup operations on its compressed representation.

Procedure compress. We represent SA succinctly by executing Steps 1–4 of phases $k = 0, 1, \dots, \ell - 1$, where the exact value of $\ell = \Theta(\lg \lg n)$ will be determined in section 3. As a result, we have $\ell + 1$ levels of information, numbered $0, 1, \dots, \ell$, which form the *compressed representation* of suffix array SA as follows:

1. Level k , for each $0 \leq k < \ell$, stores B_k , Ψ_k , and \textit{rank}_k . We do not store SA_k , but we refer to it for the sake of discussion. The arrays Ψ_k and \textit{rank}_k are not stored explicitly, but are stored in a specially compressed form described in section 3.

2. The last level $k = \ell$ stores SA_ℓ explicitly because it is sufficiently small to fit in $O(n)$ bits. The ℓ th level functionality of structures B_ℓ , Ψ_ℓ , and \textit{rank}_ℓ are not needed as a result.

Procedure lookup (i). We define $\textit{lookup}(i) = \textit{rlookup}(i, 0)$, where $\textit{rlookup}(i, k)$ is the procedure described recursively for level k in Figure 3.

If k is the last level ℓ , then it performs a direct lookup in $SA_\ell[i]$. Otherwise, it exploits Lemma 1 and the inductive hypothesis so that $\textit{rlookup}(i, k)$ returns the value of $2 \cdot SA_{k+1}[\textit{rank}_k(\Psi_k(i))] + (B_k[i] - 1)$ in $SA_k[i]$.

2.2. Compressibility. As previously mentioned, B_k , \textit{rank}_k , and Ψ_k are the key ingredients for representing a compressed suffix array. Storing B_k and \textit{rank}_k succinctly, with constant-time access, can be done using previous work (see, e.g., [42]). Hence, we focus on function Ψ_k , which is at the heart of the compressed suffix array since its compression is challenging.

Before giving some intuition on the compressibility, a few comments are in order. When $\Psi_0(i) \neq i$, we observe that Ψ_0 is the analogue of the suffix links in McCreight's

suffix tree construction [57]. (We recall that a suffix link for a node storing a nonempty string $c\alpha$, where $c \in \Sigma$, points to the node storing α .) This is clear when we consider its extension, Φ_0 , defined in section 3.2 as follows:

$$\Phi_k(i) = \begin{cases} j & \text{if } SA_k[i] \neq n_k \text{ and } SA_k[j] = SA_k[i] + 1; \\ 1 & \text{otherwise.} \end{cases}$$

Indeed, if i is the position in SA for suffix $T[SA[i], n]$, then $\Phi_0(i)$ returns j , which is the position in SA for suffix $T[SA[i] + 1, n]$ (when $\Phi_0(i)$ is seen as a suffix link, $c = T[SA[i]]$ and $\alpha = T[SA[i] + 1, n]$). Analogously, we can see Ψ_k and Φ_k as the suffix links for the positions in SA_k , for any $k \geq 0$.

Functions Ψ_k and Φ_k can be seen as by-products of the suffix array construction. Let the inverse suffix array, SA^{-1} , be the array satisfying $SA^{-1}[SA[i]] = SA[SA^{-1}[i]] = i$. Note that SA^{-1} is well defined since SA stores a permutation of $1, 2, \dots, n$. When $\Phi_0(i) \neq 1$, we have $\Phi_0(i) = SA^{-1}[SA[i] + 1]$. An analogous argument holds for any $k \geq 0$.

In order to see why Ψ_0 and Φ_0 are compressible, we focus on Ψ_0 . If we consider the values of Ψ_0 in the example of section 2.1, we do not see any particular order. However, if we restrict our focus to the positions i ($1 \leq i \leq n$) having (a) value of **0** in $B_0[i]$, and (b) the *same* leading character in the corresponding suffix, $T[SA[i], n]$, we observe that the values of Ψ_0 in those positions yield an increasing sequence. For example, choosing **a** as the leading character in condition (b), we find that the positions satisfying also condition (a) are $i = 1, 3, 4, 5, 6, 9, 11, 12$. Their corresponding values are $\Psi_0(i) = 2, 14, 15, 18, 23, 28, 30, 31$, respectively. The latter values form a sorted sequence (called **a** list) that can be implicitly represented in several ways. We clearly have to represent lists for all distinct characters that appear in the text (**a** list, **b**list, ...).

In this paper, we represent the lists by relating them to the positions of the companion **1**'s in B_0 . Let the preceding character for position i in SA be $T[SA[i] - 1]$ for $SA[i] \neq 1$; otherwise, let it be $T[n]$. We implicitly associate the preceding character for position i with each entry of B_0 containing **1**. For example, in the case of the **a** list, the positions corresponding to the **1**'s in B_0 and with preceding character **a** are $2, 14, 15, 18, 23, 28, 30, 31$, which are exactly the items in the **a** list itself! (The motivation for this nice property is that the suffixes remain sorted in relative order, even if interspersed with other suffixes, when we remove their leading character **a**.) By exploiting this relation, we can implement constant-time access to Ψ_0 's values without needing to store them explicitly. Further details on how to represent $rank_k$, Ψ_k , and Φ_k in compressed form and how to implement *compress* and *lookup(i)* will be given in section 3.

2.3. Results. Our main theorem below gives the resulting time and space complexity that we are able to achieve.

THEOREM 1 (binary alphabets). *Consider the suffix array SA built upon a binary string of length $n - 1$.*

- (i) *We can implement *compress* in $\frac{1}{2}n \lg \lg n + 6n + O(n/\lg \lg n)$ bits and $O(n)$ preprocessing time, so that each call *lookup(i)* takes $O(\lg \lg n)$ time.*
- (ii) *We can implement *compress* in $(1 + \epsilon^{-1})n + O(n/\lg \lg n)$ bits and $O(n)$ preprocessing time, so that each call *lookup(i)* takes $O(\lg^\epsilon n)$ time, for any fixed value of $0 < \epsilon \leq 1$.*

The coefficients on the second-order terms can be tweaked theoretically by a more elaborate encoding. We also state the above results in terms of alphabets with $|\Sigma| > 2$.

THEOREM 2 (general alphabets). *Consider the suffix array SA built upon a string of length $n - 1$ over the alphabet Σ with size $|\Sigma| > 2$.*

(i) *We can implement compress in $(1 + \frac{1}{2} \lg \lg_{|\Sigma|} n) n \lg |\Sigma| + 5n + O(n / \lg \lg n) = (1 + \frac{1}{2} \lg \lg_{|\Sigma|} n) n \lg |\Sigma| + O(n)$ bits and $O(n \lg |\Sigma|)$ preprocessing time, so that each call $lookup(i)$ takes $O(\lg \lg_{|\Sigma|} n)$ time.*

(ii) *We can implement compress in $(1 + \epsilon^{-1}) n \lg |\Sigma| + 2n + O(n / \lg \lg n) = (1 + \epsilon^{-1}) n \lg |\Sigma| + o(n \lg |\Sigma|)$ bits and $O(n \lg |\Sigma|)$ preprocessing time, so that each call $lookup(i)$ takes $O(\lg_{|\Sigma|}^{\epsilon} n)$ time, for any fixed value of $0 < \epsilon \leq 1$. For $|\Sigma| = O(1)$, the space bound reduces to $(1 + \epsilon^{-1}) n \lg |\Sigma| + O(n / \lg \lg n) = (1 + \epsilon^{-1}) n \lg |\Sigma| + o(n)$ bits.*

We remark that Sadakane [64] has shown that the space complexity in Theorem 1(ii) and Theorem 2(ii) can be restated in terms of the order-0 entropy $H_0 \leq \lg |\Sigma|$ of the string, giving as a result $\epsilon^{-1} H_0 n + O(n)$ bits. Grossi, Gupta, and Vitter [36] have shown how to attain order- h entropy, namely, $H_h n + O(n \lg \lg n / \lg_{|\Sigma|} n)$ bits, where $H_h \leq H_0$.

The lookup process can be sped up when we need to report several contiguous entries, as in enumerative string matching queries. Let $lcp(i, j)$ denote the length of the longest common prefix between the suffixes pointed to by $SA[i]$ and $SA[j]$, with the convention that $lcp(i, j) = -\infty$ when $i < 1$ or $j > n$. We say that a sequence $i, i + 1, \dots, j$ of indices in SA is *maximal* if both $lcp(i - 1, j)$ and $lcp(i, j + 1)$ are strictly smaller than $lcp(i, j)$, as in enumerative queries. (Intuitively, a maximal sequence in SA corresponds to all the occurrences of a pattern in T .)

THEOREM 3 (batch of lookups). *In each of the cases stated in Theorems 1 and 2, we can use the additional space of $O(n \lg |\Sigma|)$ bits and batch together $j - i + 1$ procedure calls $lookup(i), lookup(i + 1), \dots, lookup(j)$, for a maximal sequence $i, i + 1, \dots, j$, so that the total cost is*

(i) *$O(j - i + (\lg n)^{1+\epsilon} (\lg |\Sigma| + \lg \lg n))$ time when $lcp(i, j) = \Omega(\lg^{1+\epsilon} n)$, namely, the suffixes pointed to by $SA[i]$ and $SA[j]$ have the same first $\Omega(\lg^{1+\epsilon} n)$ symbols in common, or*

(ii) *$O(j - i + n^{\alpha})$ time, for any constant $0 < \alpha < 1$, when $lcp(i, j) = \Omega(\lg n)$, namely, the suffixes pointed to by $SA[i]$ and $SA[j]$ have the same first $\Omega(\lg n)$ symbols.*

3. Algorithms for compressed suffix arrays. In this section we constructively prove Theorems 1–3 by showing two ways to implement the recursive decomposition of suffix arrays discussed in section 2.1. In particular, in section 3.1 we address Theorem 1(i), and in section 3.2 we prove Theorem 1(ii). Section 3.3 shows how to extend Theorem 1 to deal with alphabets of size $|\Sigma| > 2$, thus proving Theorem 2. In section 3.4 we prove Theorem 3, showing how to batch together the lookup of several contiguous entries in suffix arrays, which arises in enumerative string matching queries.

3.1. Compressed suffix arrays in $\frac{1}{2} n \lg \lg n + O(n)$ bits and $O(\lg \lg n)$ access time. In this section we describe the method referenced in Theorem 1(i) for binary strings and show that it achieves $O(\lg \lg n)$ lookup time with a total space usage of $O(n \lg \lg n)$ bits. Before giving the algorithmic details of the method, let's continue the recursive decomposition of Steps 1–4 described in section 2.1, for $0 \leq k \leq \ell - 1$, where $\ell = \lceil \lg \lg n \rceil$. The decomposition below shows the result on the example of section 2.1:

	1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16	
SA_1 :	8 14 5 2 12 16 7 15 6 9 3 10 13 4 1 11	
B_1 :	1 1 0 1 1 1 0 0 1 0 0 1 0 1 0 0	
$rank_1$:	1 2 2 3 4 5 5 5 6 6 6 7 7 8 8 8	
Ψ_1 :	1 2 9 4 5 6 1 6 9 12 14 12 2 14 4 5	
	1 2 3 4 5 6 7 8	
SA_2 :	4 7 1 6 8 3 5 2	
B_2 :	1 0 0 1 1 0 0 1	1 2 3 4
$rank_2$:	1 1 1 2 3 3 3 4	SA_3 : 2 3 4 1
Ψ_2 :	1 5 8 4 5 1 4 9	

The resulting suffix array SA_ℓ on level ℓ contains at most $n/\lg n$ entries and can thus be stored explicitly in at most n bits. We store the bit vectors $B_0, B_1, \dots, B_{\ell-1}$ in explicit form, using less than $2n$ bits, as well as *implicit* representations of $rank_0, rank_1, \dots, rank_{\ell-1}$, and $\Psi_0, \Psi_1, \dots, \Psi_{\ell-1}$. If the implicit representations of $rank_k$ and Ψ_k can be accessed in constant time, the procedure described in Lemma 1 shows how to achieve the desired lookup in constant time per level, for a total of $O(\lg \lg n)$ time.

All that remains, for $0 \leq k \leq \ell - 1$, is to investigate how to represent $rank_k$ and Ψ_k in $O(n)$ bits and support constant-time access. Given the bit vector B_k of $n_k = n/2^k$ bits, Jacobson [42] shows how to support constant-time access to $rank_k$ using only $O(n_k(\lg \lg n_k)/\lg n_k)$ extra bits, with preprocessing time $O(n_k)$.

We show next how to represent Ψ_k implicitly. First we explain the representation by an example and then we describe it formally. In Lemma 3 we show that the space used to represent Ψ_k is $n(1/2 + 3/2^{k+1}) + O(n/2^k \lg \lg n)$ bits.

For each $1 \leq i \leq n_k/2$, let j be the index of the i th **1** in B_k . Consider the 2^k symbols in positions $2^k \cdot (SA_k[j] - 1), \dots, 2^k \cdot SA_k[j] - 1$ of T ; these 2^k symbols immediately precede the $(2^k \cdot SA_k[j])$ th suffix in T , as the suffix pointer in $SA_k[j]$ was 2^k times larger before the compression. For each bit pattern of 2^k symbols that appears, we keep an ordered list of the indices $j \in [1, n/2^k]$ that correspond to it, and we record the number of items in each list. Continuing the example above, we get the following lists for level 0:

a list:	$\langle 2, 14, 15, 18, 23, 28, 30, 31 \rangle$,	$ a \text{ list} = 8$
b list:	$\langle 7, 8, 10, 13, 16, 17, 21, 27 \rangle$,	$ b \text{ list} = 8$

Level 1:

aa list:	\emptyset ,	$ aa \text{ list} = 0$
ab list:	$\langle 9 \rangle$,	$ ab \text{ list} = 1$
ba list:	$\langle 1, 6, 12, 14 \rangle$,	$ ba \text{ list} = 4$
bb list:	$\langle 2, 4, 5 \rangle$,	$ bb \text{ list} = 3$

Level 2:

aaaa list:	\emptyset ,	$ aaaa \text{ list} = 0$	baaa list:	\emptyset ,	$ baaa \text{ list} = 0$
aaab list:	\emptyset ,	$ aaab \text{ list} = 0$	baab list:	\emptyset ,	$ baab \text{ list} = 0$
aaba list:	\emptyset ,	$ aaba \text{ list} = 0$	baba list:	$\langle 1 \rangle$,	$ baba \text{ list} = 1$
aabb list:	\emptyset ,	$ aabb \text{ list} = 0$	babb list:	$\langle 4 \rangle$,	$ babb \text{ list} = 1$
abaa list:	\emptyset ,	$ abaa \text{ list} = 0$	bbaa list:	\emptyset ,	$ bbaa \text{ list} = 0$
abab list:	\emptyset ,	$ abab \text{ list} = 0$	bbab list:	\emptyset ,	$ bbab \text{ list} = 0$
abba list:	$\langle 5, 8 \rangle$,	$ abba \text{ list} = 2$	bbba list:	\emptyset ,	$ bbba \text{ list} = 0$
abbb list:	\emptyset ,	$ abbb \text{ list} = 0$	bbbb list:	\emptyset ,	$ bbbb \text{ list} = 0$

Suppose we want to compute $\Psi_k(i)$. If $B_k[i] = \mathbf{1}$, we trivially have $\Psi_k(i) = i$; therefore, let's consider the harder case in which $B_k[i] = \mathbf{0}$, which means that $SA_k[i]$ is odd. We have to determine the index j such that $SA_k[j] = SA_k[i] + 1$. We can determine the number h of $\mathbf{0}$'s in B_k up to index i by computing $i - \text{rank}_k(i)$, i.e., by subtracting the number of $\mathbf{1}$'s in the first i bits of B_k . Consider the 2^{2^k} lists concatenated together in lexicographic order of the 2^k -bit prefixes. We denote by L_k the resulting concatenated list, which has $|L_k| = n_k/2 = n/2^{k+1}$ total items. What we need to find now is the h th entry in L_k . For example, to determine $\Psi_0(25)$ in the example above, we find that there are $h = 13$ $\mathbf{0}$'s in the first 25 slots of B_0 . There are eight entries in the **a** list and eight entries in the **b** list; hence, the 13th entry in L_0 is the fifth entry in the **b** list, namely, index 16. Hence, we have $\Psi_0(25) = 16$ as desired; note that $SA_0[25] = 29$ and $SA_0[16] = 30$ are consecutive values.

Continuing the example, consider the next level of the recursive call to *rlookup*, in which we need to determine $\Psi_1(8)$. (The previously computed value $\Psi_0(25) = 16$ has a rank_0 value of 8, i.e., $\text{rank}_0(16) = 8$, so the *rlookup* procedure needs to determine $SA_1[8]$, which it does by first calculating $\Psi_1(8)$.) There are $h = 3$ $\mathbf{0}$'s in the first eight entries of B_1 . The third entry in the concatenated list L_1 for **aa**, **ab**, **ba**, and **bb** is the second entry in the **ba** list, namely, 6. Hence, we have $\Psi_1(8) = 6$ as desired; note that $SA_1[8] = 15$ and $SA_1[6] = 16$ are consecutive values.

We now describe formally how to preprocess the input text T in order to form the concatenated list L_k on level k used for Ψ_k with the desired space and constant-time query performance. We first consider a variant of the "inventories" introduced by Elias [23] to get average bit efficiency in storing sorted multisets. We show how to get worst-case efficiency.

LEMMA 2 (constant-time access to compressed sorted integers). *Given s integers in sorted order, each containing w bits, where $s < 2^w$, we can store them with at most $s(2 + w - \lfloor \lg s \rfloor) + O(s/\lg \lg s)$ bits, so that retrieving the h th integer takes constant time.*

Proof. We take the first $z = \lfloor \lg s \rfloor$ bits of each integer in the sorted sequence. Let q_1, \dots, q_s be the integers so obtained, called *quotients*, where $0 \leq q_h \leq q_{h+1} < s$ for $1 \leq h < s$. (Note that multiple values are allowed.) Let r_1, \dots, r_s be the *remainders*, obtained by deleting the first z bits from each integer in the sorted sequence.

We store q_1, \dots, q_s in a table Q described below, requiring $2s + O(s/\lg \lg s)$ bits. We store r_1, \dots, r_s in a table R taking $s(w - z)$ bits. Table R is the simple concatenation of the bits representing r_1, \dots, r_s .

As for Q , we use the unary representation $\mathbf{0}^i \mathbf{1}$ (i.e., i copies of $\mathbf{0}$ followed by $\mathbf{1}$) to represent integer $i \geq 0$. Then we take the concatenation of the unary representation of $q_1, q_2 - q_1, \dots, q_s - q_{s-1}$. In other words, we take the first entry encoded in unary, and then the unary difference between the other consecutive entries, which are in nondecreasing order. Table Q is made up of the binary string obtained by the above concatenation S , augmented with the auxiliary data structure supporting *select* operations to locate the position of the h th $\mathbf{1}$ in constant time [15, 42, 60].

Since S requires $s + 2^z \leq 2s$ bits, the total space required by Q is $2s + O(s/\lg \lg s)$ bits; the big-oh term is due to the auxiliary data structure that implements *select*. In order to retrieve q_h , we find the position j of the h th $\mathbf{1}$ in S by calling *select*(h), and then compute the number of $\mathbf{0}$'s in the first j bits of S by returning $j - h$. As we can see, this number of $\mathbf{0}$'s gives q_h . The time complexity is constant.

In order to obtain the h th integer in the original sorted sequence, we find q_h by querying Q as described above, and we find r_i by looking up the h th entry in R .

We then output $q_h \cdot 2^{w-z} + r_h$ as the requested integer by simply returning the concatenation of the bit representations of q_h and r_h . \square

We now proceed to the implementation of Ψ_k .

LEMMA 3. *We can store the concatenated list L_k used for Ψ_k in $n(1/2+3/2^{k+1}) + O(n/2^k \lg \lg n)$ bits, so that accessing the h th entry in L_k takes constant time. Pre-processing time is $O(n/2^k + 2^{2^k})$.*

Proof. There are $d = 2^{2^k}$ lists, some of which may be empty. We number the lists composing L_k from 0 to $2^{2^k} - 1$. Each integer x in list i , where $1 \leq x \leq n_k$, is transformed into an integer x' of $w = 2^k + \lg n_k$ bits by prepending the binary representation of i to that of $x - 1$. Given any such x' , we can obtain the corresponding x in constant time. As a result, L_k contains $s = n_k/2 = n/2^{k+1}$ integers in increasing order, each integer of w bits. By Lemma 2, we can store L_k in $s(2 + w - \lg s) + O(s/\lg \lg s)$ bits, so that retrieving the h th integer takes constant time. Substituting the values for s and w , we get the space bound $(n_k/2)(2 + 2^k + \lg n_k - \lg(n_k/2)) + O(n_k/\lg \lg n_k) = (n/2^{k+1})(2^k + 3) + O(n/2^k \lg \lg n) = n(1/2 + 3/2^{k+1}) + O(n/2^k \lg \lg n)$. \square

A good way to appreciate the utility of the data structure for Ψ_k is to consider the naive alternative. Imagine that the information is stored naively in the form of an unsorted array of $s = n_k/2$ entries, where each entry specifies the particular list to which the entry belongs. Since there are $d = 2^{2^k}$ lists, the total number of bits needed to store the array in this naive manner is $s \lg d = (n_k/2)2^k = n/2$, which is efficient in terms of space. Let us define the natural ordering \prec on the array entries, in which we say that $i \prec j$ either if $i < j$ or if $i = j$ and the position of i in the array precedes the position of j . The naive representation does not allow us to efficiently look up the h th \prec -ordered entry in the array, which is equivalent to finding the h th entry in the concatenated list L_k . It also doesn't allow us to search quickly for the g th occurrence of the entry i , which is equivalent to finding the g th item in list i . In contrast, the data structure described in Lemma 3 supports both of these query operations in linear space and constant time.

COROLLARY 1. *Given an unsorted array of s entries, each in the range $[0, d - 1]$, we can represent the array in a total of $s \lg d + O(s)$ bits so that, given h , we can find the h th entry (in \prec order) in the array in constant time. We can also represent the array in $O(s \lg d)$ bits so that, given g and i , we can find the g th occurrence of i in the array in constant time. The latter operation can be viewed as a generalization of the select operation to arbitrary input patterns.*

Proof. The first type of query is identical to finding the h th item in the concatenated list L_k , and the bound on space follows from the construction in Lemma 3. The corresponding values of s and w in the proof of Lemma 3 are s and $\lg d + \lg s$, respectively.

The second type of query is identical to finding the g th entry in list i . It can be turned into the first type of query if we can compute the value of h that corresponds to g and i ; that is, we need to find the global position h (with respect to \prec) of the g th entry in list i . If $d \leq s$, then we can explicitly store a table that gives for each $0 \leq i < d$ the first location h' in the concatenated list L_k that corresponds to an entry in list i . We then set $h = h' + g - 1$ and do the first type of query. (If list i has fewer than g entries, which can be detected after the query is done, the value returned by the first query must be nullified.) The total space used is $d \lg s$, which by the assumption $d \leq s$ is at most $s \lg d$. If instead $d > s$, then we can use the same approach as above, except that we substitute a perfect hash function to compute the value h' . The space for the hash table is $O(s \lg s) = O(s \lg d)$. \square

Putting it all together. At this point, we have all the pieces needed to finish the proof of Theorem 1(i). Given text T and its suffix array, we proceed in $\ell = \lceil \lg \lg n \rceil$ levels of decomposition as discussed in procedure *compress* in section 2. The last level ℓ stores explicitly a reduced suffix array in $(n/2^\ell) \lg n \leq n$ bits. The other levels $0 \leq k \leq \ell - 1$ store three data structures each, with constant time access as follows:

1. Bit vector B_k of size $n_k = n/2^k$, with $O(n_k)$ preprocessing time.
2. Function $rank_k$ in $O(n_k(\lg \lg n_k)/\lg n_k)$ bits, with $O(n_k)$ preprocessing time.
3. Function Ψ_k in $n(1/2 + 3/2^{k+1}) + O(n/2^k \lg \lg n)$ bits, with $O(n_k + 2^{2^k})$ preprocessing time (see Lemma 3).

By summing over the levels, substituting the values $\ell = \lceil \lg \lg n \rceil$ and $n_k = n/2^k$, we get the following bound on the total space:

$$\begin{aligned}
 & \frac{n \lg n}{2^\ell} + \sum_{k=0}^{\ell-1} n \left(\frac{1}{2^k} + O\left(\frac{1}{2^k} \frac{\lg \lg(n/2^k)}{\lg(n/2^k)}\right) + \frac{1}{2} + \frac{3}{2^{k+1}} + O\left(\frac{1}{2^k \lg \lg n}\right) \right) \\
 & < \frac{n \lg n}{2^\ell} + n \left(2 + O\left(\frac{\lg \lg n}{\lg n}\right) + \frac{1}{2} \ell + 3 + O\left(\frac{1}{\lg \lg n}\right) \right) \\
 (1) \quad & = \frac{n \lg n}{2^\ell} + \frac{1}{2} \ell n + 5n + O\left(\frac{n}{\lg \lg n}\right).
 \end{aligned}$$

It's easy to show that $(n \lg n)/2^\ell + \frac{1}{2} \ell n \leq \frac{1}{2} n \lg \lg n + n$, which combined with (1) gives us the desired space bound $\frac{1}{2} n \lg \lg n + 6n + O(n/\lg \lg n)$ in Theorem 1(i).

The total preprocessing time of *compress* is $\sum_{k=0}^{\ell-1} O(n_k + 2^{2^k}) = O(n)$. A call to *lookup* goes through the $\ell + 1$ levels, in constant time per level, with a total cost of $O(\lg \lg n)$. This completes the proof of Theorem 1(i).

3.2. Compressed suffix arrays in $\epsilon^{-1}n + O(n)$ bits and $O(\lg^\epsilon n)$ access time. In this section we give the proof of Theorem 1(ii). Each of the $\lceil \lg \lg n \rceil$ levels of the data structure discussed in the previous section 3.1 uses $O(n)$ bits, so one way to reduce the space complexity is to store only a constant number of levels, at the cost of increased access time. For example, we can keep a total of three levels: level 0, level ℓ' , and level ℓ , where $\ell' = \lceil \frac{1}{2} \lg \lg n \rceil$ and, as before, $\ell = \lceil \lg \lg n \rceil$. In the previous example of $n = 32$, the three levels chosen are levels 0, 2, and 3. The trick is to determine how to reconstruct SA_0 from $SA_{\ell'}$ and how to reconstruct $SA_{\ell'}$ from SA_ℓ .

We store the $n_{\ell'}$ indices from SA_0 that correspond to the entries of $SA_{\ell'}$ in a new dictionary D_0 , and similarly we store the n_ℓ indices from $SA_{\ell'}$ that correspond to the entries of SA_ℓ in a new dictionary $D_{\ell'}$. By using the efficient static dictionary representation in [13, 63], we need less than $O(\lg \binom{n}{n_{\ell'}}) = O(n_{\ell'} \ell')$ bits for D_0 and $O(\lg \binom{n_{\ell'}}{n_\ell}) = O(n_\ell \ell)$ bits for $D_{\ell'}$. A dictionary lookup requires constant time, as does a rank query to know how many smaller or equal indices are stored in the dictionary [63].

We also have a data structure for $k = 0$ and $k = \ell'$ to support the function Ψ'_k , which is similar to Ψ_k , except that it maps $\mathbf{1}$'s to the next corresponding $\mathbf{0}$. We denote by Φ_k the resulting composition of Ψ_k and Ψ'_k , for $1 \leq i \leq n_k$:

$$\Phi_k(i) = \begin{cases} j & \text{if } SA_k[i] \neq n_k \text{ and } SA_k[j] = SA_k[i] + 1; \\ 1 & \text{otherwise.} \end{cases}$$

We implement Φ_k by merging the concatenated lists L_k of Ψ_k with the concatenated lists L'_k of Ψ'_k . For example, in level $k = 0$ shown in section 3.1, we merge the a list

of L_k with the \mathbf{a} list of L'_k , and so on (we need also the singleton list for $\#$). This is better than storing L_k and L'_k separately. Computing $\Phi_k(i)$ amounts to taking the i th entry in its concatenated list, and we no longer need the bit vector B_k .

LEMMA 4. *We can store the concatenated lists used for Φ_k in $n + O(n/\lg \lg n)$ bits for $k = 0$, and $n(1 + 1/2^{k-1}) + O(n/2^k \lg \lg n)$ bits for $k > 0$, so that accessing the h th entry takes constant time. Preprocessing time is $O(n/2^k + 2^{2^k})$.*

Proof. For $k > 0$, the proof is identical to that of Lemma 3, except that $s = n_k$ instead of $s = n_k/2$. For $k = 0$, we have only the \mathbf{a} list and the \mathbf{b} list to store, with the singleton $\#$ list handled a bit differently. Specifically, we encode \mathbf{a} and $\#$ by $\mathbf{0}$ and \mathbf{b} by $\mathbf{1}$. Then, we create a bit vector of n bits, where the bit in position f is $\mathbf{0}$ if the list for Φ_0 contains either \mathbf{a} or $\#$ in position f , and it is $\mathbf{1}$ if it contains \mathbf{b} in position f . We use auxiliary information to access the i th $\mathbf{1}$ of the bit vector in constant time by using $select(i)$ or the i th $\mathbf{0}$ by using $select_0(i)$. We also keep a counter c_0 to know the total number of $\mathbf{0}$'s in the bit vector (note that the single occurrence of $\mathbf{0}$ corresponding to $\#$ in the bit vector is the c_0 th $\mathbf{0}$ in the bit vector as we assumed $\mathbf{a} < \# < \mathbf{b}$; it is not difficult to treat the more common case $\# < \mathbf{a} < \mathbf{b}$). The additional space is $O(n/\lg \lg n)$ due to the implementation of $select$ and $select_0$. Suppose now that we want to recover the h th entry in the list for Φ_0 . If $h = c_0$, then we return the position of $\#$ by invoking $select_0(c_0)$. If $h < c_0$, then we return the h th $\mathbf{0}$ (i.e., \mathbf{a}) in the bit vector by invoking $select_0(h)$. Otherwise, we invoke $select(h - c_0)$ to get the position in the bit vector of the $(h - c_0)$ th $\mathbf{1}$ (i.e., \mathbf{b}). In this way, we simulate the concatenation of lists needed for L_0 . With $n + O(n/\lg \lg n)$ bits to implement Φ_0 , we can execute $\Phi_0(h)$ in constant time. \square

In order to determine $SA[i] = SA_0[i]$, we use function Φ_0 to walk along indices i', i'', \dots , such that $SA_0[i] + 1 = SA_0[i']$, $SA_0[i'] + 1 = SA_0[i'']$, and so on, until we reach an index stored in dictionary D_0 . Let s be the number of steps in the walk and r be the rank of the index thus found in D_0 . We switch to level ℓ' and reconstruct the r th entry at level ℓ' from the explicit representation of $SA_{\ell'}$ by a similar walk until we find an index stored in $D_{\ell'}$. Let s' be the number of steps in the latter walk and r' be the rank of the index thus found in $D_{\ell'}$. We return $(SA_{\ell'}[r'] \cdot 2^{\ell} + s' \cdot 2^{\ell'} + s \cdot 2^0)$, as this is the value of $SA_0[i]$. We defer details for reasons of brevity. The maximum length of each walk is $\max\{s, s'\} \leq 2^{\ell'} < 2\sqrt{\lg n}$, and thus the lookup procedure requires $O(\sqrt{\lg n})$ time.

To get the more general result stated in Theorem 1(ii), we need to keep a total of $\epsilon^{-1} + 1$ levels for constant $0 < \epsilon \leq 1$. More formally, let us assume that $\epsilon \ell$ is an integer. We maintain the $\epsilon^{-1} + 1$ levels $0, \epsilon \ell, 2\epsilon \ell, \dots, \ell$. The maximum length of each walk is $2^{\epsilon \ell} < 2 \lg^{\epsilon} n$, and thus the lookup procedure requires $O(\lg^{\epsilon} n)$ time.

By an analysis similar to the one we used at the end of section 3.1, the total space bound is given by $(n/2^{\ell}) \lg n \leq n$ plus a sum over the ϵ^{-1} indices $k \in \{0, \epsilon \ell, 2\epsilon \ell, 3\epsilon \ell, \dots, (1 - \epsilon)\ell\}$. We split the sum into two parts, one for $k = 0$ and the other for the remaining $\epsilon^{-1} - 1$ values of $k > 0$, and apply Lemma 4:

$$\begin{aligned}
 & \frac{n \lg n}{2^{\ell}} + n + O\left(\frac{n}{\lg \lg n}\right) + \sum_{\substack{k=i\epsilon\ell \\ 1 \leq i < \epsilon^{-1}}} n \left(1 + \frac{1}{2^{k-1}} + O\left(\frac{1}{2^k \lg \lg n}\right)\right) \\
 & \leq (1 + \epsilon^{-1})n + O\left(\frac{n}{\lg \lg n}\right) + O\left(\frac{n}{\lg^{\epsilon} n}\right) \\
 (2) \quad & = (1 + \epsilon^{-1})n + O\left(\frac{n}{\lg \lg n}\right).
 \end{aligned}$$

We have to add the contribution of the space $\sum_k |D_k| = O(n_{\epsilon\ell}) = O(n(\lg \lg n)/\lg^\epsilon n)$ taken by the dictionaries at the ϵ^{-1} levels, but this bound is hidden by the $O(n/\lg \lg n)$ term in the above formula. The final bound is $(1 + \epsilon^{-1})n + O(n/\lg \lg n)$, as stated in Theorem 1(ii).

3.3. Extension to alphabets of size $|\Sigma| > 2$. We now discuss the case of alphabets with more than two symbols. In this case, we encode each symbol by $\lg |\Sigma|$ bits, so that the text T can be seen as an array of n entries, each of $\lg |\Sigma|$ bits, or equivalently as a binary string that occupies $n \lg |\Sigma|$ bits. We describe how to extend the ideas presented in sections 3.1–3.2. We redefine ℓ to be $\lceil \lg \lg_{|\Sigma|} n \rceil$. The definitions of suffix arrays SA and SA_k , bit vector B_k , and functions $rank_k$ and Ψ_k are the same as before. Their representation does not change, with the notable exception of Ψ_k , as noted below in Lemma 5 (the analogue of Lemma 3).

LEMMA 5. *When $|\Sigma| > 2$, we can store the concatenated list L_k used for Ψ_k in $n((1/2) \lg |\Sigma| + 3/2^{k+1}) + O(n/2^k \lg \lg n)$ bits, so that accessing the h th entry in L_k takes constant time. Preprocessing time is $O(n/2^k + 2^{2^k})$.*

Proof. The extension of L_k with $|\Sigma| > 2$ is straightforward. For each of $d = |\Sigma|^{2^k} = 2^{2^k \lg |\Sigma|}$ patterns of 2^k symbols preceding the $(2^k \cdot SA_k[j])$ th suffix in T , we keep an ordered list like the **a** list and **b** list described in section 3.1. Some of these lists may be empty, and the concatenation of nonempty lists forms L_k . We number these lists from 0 to $2^{2^k \lg |\Sigma|} - 1$. Note that the number of entries in L_k remains unchanged, namely, $s = n_k/2 = n/2^{k+1}$. Each integer x in list i , where $1 \leq x \leq n_k$, is transformed into an integer x' of $w = 2^k \lg |\Sigma| + \lg n_k$ bits, by prepending the binary representation of i to that of $x - 1$. By Lemma 2, we can store L_k in $s(2 + w - \lg s) + O(s/\lg \lg s)$ bits, so that retrieving the h th integer takes constant time. Substituting the values for s and w , we get the space bound $(n_k/2)(2 + 2^k \lg |\Sigma| + \lg n_k - \lg(n_k/2)) + O(n_k/\lg \lg n_k) = n((1/2) \lg |\Sigma| + 3/2^{k+1}) + O(n/2^k \lg \lg n)$ \square

By replacing the space complexity of Ψ_k in formula (1) at the end of section 3.1, we obtain

$$\begin{aligned} & \frac{n \lg n}{2^\ell} + \sum_{k=0}^{\ell-1} n \left(\frac{1}{2^k} + O\left(\frac{1}{2^k} \frac{\lg \lg(n/2^k)}{\lg(n/2^k)}\right) + \frac{\lg |\Sigma|}{2} + \frac{3}{2^{k+1}} + O\left(\frac{1}{2^k \lg \lg n}\right) \right) \\ & < \left(1 + \frac{1}{2} \lg \lg_{|\Sigma|} n\right) n \lg |\Sigma| + 5n + O\left(\frac{n}{\lg \lg n}\right), \end{aligned}$$

as $(n \lg n)/2^\ell + \frac{1}{2} \ell n \leq (1 + \frac{1}{2} \lg \lg_{|\Sigma|} n) n \lg |\Sigma|$, thus proving Theorem 2(i).

To prove Theorem 2(ii), we follow the approach of section 3.2. We need dictionaries D_k and functions Φ_k for $k \in \{0, \epsilon\ell, 2\epsilon\ell, 3\epsilon\ell, \dots, (1 - \epsilon)\ell\}$. Their definitions and representations do not change, except for the representation of Φ_k , for which we need Lemma 6 (the analogue of Lemma 4).

LEMMA 6. *We can store the concatenated lists used for Φ_k in $n(\lg |\Sigma| + 1/2^{k-1}) + O(n/2^k \lg \lg n)$ bits, so that accessing the h th entry takes constant time. Preprocessing time is $O(n/2^k + 2^{2^k})$. When $|\Sigma| = O(1)$ and $k = 0$, we can store Φ_k in $n \lg |\Sigma| + O(|\Sigma|n/\lg \lg n) = n \lg |\Sigma| + o(n)$ bits.*

Proof. The proof is identical to that of Lemma 5, except that $s = n_k$ instead of $s = n_k/2$. When $|\Sigma| = O(1)$, we can use a better approach for $k = 0$ as in the proof of Lemma 4. We associate $\lg |\Sigma|$ bits with each character in Σ according to its lexicographic order. Then we use a bit vector of $n \lg |\Sigma|$ bits to represent Φ_0 , in which the f th chunk of $\lg |\Sigma|$ bits encoding a character $c \in \Sigma$ represents the fact that the

c list for Φ_0 contains position f . We then implement $|\Sigma| = O(1)$ versions of *select*, one version per character of Σ . The version for $c \in \Sigma$ is in charge of selecting the i th occurrence of c encoded in binary in the bit vector. To this end, it treats each occurrence of the $\lg |\Sigma|$ bits for c in the bit vector as a single $\mathbf{1}$ and the occurrences of the rest of the characters as single $\mathbf{0}$'s. It should be clear that the implementation of each version of *select* can be done in $O(n/\lg \lg n)$ bits. To execute $\Phi_0(h)$ in constant time, we proceed as in Lemma 4, generalized to more than two characters. \square

By an analysis similar to the one we used in formula (2) at the end of section 3.2, we obtain

$$\begin{aligned} \frac{n \lg n}{2^\ell} + \sum_{\substack{k=i\epsilon\ell \\ 0 \leq i < \epsilon^{-1}}} n \left(\lg |\Sigma| + \frac{1}{2^{k-1}} + O\left(\frac{1}{2^k \lg \lg n}\right) \right) \\ \leq (1 + \epsilon^{-1}) n \lg |\Sigma| + 2n + O\left(\frac{n}{\lg \lg n}\right). \end{aligned}$$

When $|\Sigma| = O(1)$, we can split the above sum for $k = 0$ and apply Lemma 6 to get $(1 + \epsilon^{-1}) n \lg |\Sigma| + O(n/\lg \lg n)$ bits, thus proving Theorem 2(ii).

3.4. Output-sensitive reporting of multiple occurrences. In this section we prove Theorem 3 by showing how to output a contiguous set $SA_0[i], \dots, SA_0[j]$ of entries from the compressed suffix array under the hypothesis that the sequence $i, i + 1, \dots, j$ is maximal (according to the definition given before Theorem 3) and the corresponding suffixes share at least a certain number of initial symbols. This requires adding further $O(n \lg |\Sigma|)$ bits of space to the compressed suffix array. One way to output the $j - i + 1$ entries is via a reduction to two-dimensional orthogonal range search [46]. Let D be a two-dimensional orthogonal range query data structure on q points in the grid space $[1 \dots U] \times [1 \dots U]$, where $1 \leq q \leq U$. Let $P(q)$ be its preprocessing time, $S(q)$ the number of occupied words of $O(\lg U)$ bits each, and $T(q) + O(k)$ the cost of searching and retrieving the k points satisfying a given range query in D .

LEMMA 7. Fix $U = n$ in the range query data structure D , and let $n' \geq 1$ be the largest integer such that $S(n') = O(n/\lg n)$. If such an n' exists, we can report $SA[i], \dots, SA[j]$ in $O(\lg_{|\Sigma|}^{1+\epsilon} n + (n/n')(T(n') + \lg |\Sigma|) + j - i)$ time when the sequence $i, i + 1, \dots, j$ is maximal and the suffixes pointed to by $SA[i], \dots, SA[j]$ have the same first $\Omega(n/n')$ symbols in common. Preprocessing time is $P(n') + O(n \lg |\Sigma|)$ and space is $O(n \lg |\Sigma|)$ bits in addition to that of the compressed version of SA .

Proof. Suppose by hypothesis that the suffixes pointed to by $SA[i], \dots, SA[j]$ have in common at least $l = \lceil n/n' \rceil$ symbols. (This requirement can be further reduced to $l = \Theta(n/n')$.) We denote these symbols by b_0, b_1, \dots, b_{l-1} , from left to right.

In order to define the two-dimensional points in D , we need to build the compressed version of the suffix array SA^R for the reversal of the text, denoted T^R . Then we obtain the points to keep in D by processing the suffix pointers in SA that are multiples of l (i.e., they refer to the suffixes in T starting at positions $l, 2l, 3l, \dots$). Specifically, the point corresponding to pointer $p = SA[s]$, where $1 \leq s \leq n$ and p is a multiple of l , has first coordinate s . Its second coordinate is given by the position r of $(T[1, p - 1])^R$ in the sorted order induced by SA^R . In other words, s is the rank of $T[p, n]$ among the suffixes of T in lexicographic order, and r is the rank of $(T[1, p - 1])^R$ among the suffixes of T^R (or, equivalently, the reversed prefixes of T). Point $\langle s, r \rangle$ corresponding to p has label p to keep track of this correspondence.

Since there are $q \leq n'$ such points stored in D and we build the compressed suffix array of T^R according to Theorem 2(ii), space is $S(n') \cdot O(\lg n) + (\epsilon^{-1} + O(1)) n \lg |\Sigma| = O(n \lg |\Sigma|)$ bits. Preprocessing time is $P(n') + O(n \lg |\Sigma|)$.

We now describe how to query D and output $SA[i], \dots, SA[j]$ in l stages, with one range query per stage. In stage 0, we perform a range query for the points in $[i \dots j] \times [1 \dots n]$. For these points, we output the suffix pointers labeling them. Then we locate the leftmost suffix and the rightmost suffix in SA^R starting with $b_{l-1} \dots b_1 b_0$. For this purpose, we run a simple binary search in the compressed version of SA^R , comparing at most $\lg n$ bits at a time. As a result, we determine two positions g and h of SA^R in $O(l \lg |\Sigma| + \lg_{|\Sigma|}^{1+\epsilon} n)$ time such that the sequence $g, g + 1, \dots, h$ is maximal for SA^R and the suffixes of T^R pointed to by $SA^R[g], \dots, SA^R[h]$ start with $b_{l-1} \dots b_1 b_0$.

Before proceeding with the next stages, we precompute some sequences of indices starting from i, j, g , and h , respectively, as done in section 3.2. We use the function Φ_0 in the compressed version of $SA = SA_0$ to walk along indices i_0, i_1, \dots, i_{l-1} , such that $i_0 = i, SA_0[i_0] + 1 = SA_0[i_1], SA_0[i_1] + 1 = SA_0[i_2]$, and so on. An analogous walk applies to $j_0 = j, j_1, \dots, j_{l-1}$. In the same way, we use the function Φ_0 in the compressed version of SA^R to obtain $g_0 = g, g_1, \dots, g_{l-1}$ and $h_0 = h, h_1, \dots, h_{l-1}$. We then run the t th stage, for $1 \leq t \leq l - 1$, in which we perform a range query for the points in $[i_t \dots j_t] \times [g_{l-t} \dots h_{l-t}]$. For each of these points, we retrieve its label p and output $p - t$.

In order to see why the above method works, let us consider an arbitrary suffix pointer in $SA[i], \dots, SA[j]$. By the definition of the points kept in D , this suffix pointer can be written as $p - t$, where p is the nearest multiple of l and $0 \leq t \leq l - 1$. We show that we output $p - t$ correctly in stage t . Let $\langle s, r \rangle$ be the point with label p in D . We have to show that $i_t \leq s \leq j_t$ and $g_{l-t} \leq r \leq h_{l-t}$ (setting border values $g_l = 1$ and $h_l = n$). Recall that the suffixes pointed to by $SA[i], p - t$ and $SA[j]$ are in lexicographic order by definition of the (compressed) suffix array and, moreover, they share the first l symbols. If we remove the first $t < l$ symbols from each of them, the lexicographic order must be preserved because these symbols are equal. Consequently, $SA[i] - t, p$, and $SA[j] - t$ are still in lexicographic order, and their ranks are i_h, s , and j_h , respectively. This implies $i_t \leq s \leq j_t$. A similar property holds for $g_{l-t} \leq r \leq h_{l-t}$, and we can conclude that p is retrieved in stage t giving $p - t$ as output. Finally, the fact that both $i, i + 1, \dots, j$ and $g, g + 1, \dots, h$ are maximal sequences in their respective suffix arrays implies that no other suffix pointers besides those in $SA[i], \dots, SA[j]$ are reported.

The cost of each stage is $T(n')$ plus the output-sensitive cost of the reported suffix pointers. Stage 0 requires an additional cost of $O((n/n') \lg |\Sigma| + \lg^{1+\epsilon} n)$ to compute g and h , and a cost of $O(n/n')$ to precompute the four sequences of indices, because the length of the walks is l . The total time complexity is therefore $O((n/n')(T(n') + \lg |\Sigma|) + \lg^{1+\epsilon} n + j - i)$, where $O(j - i + 1)$ is the sum of the output-sensitive costs for reporting all the suffix pointers. \square

We use Lemma 7 to prove Theorem 3. We employ two range query data structures for D . The first one in [1] takes $P(q) = O(q \lg q)$ preprocessing time by using the perfect hash in [39], which has constant lookup time and takes $O(q \lg q)$ construction time. Space is $S(q) = O(q \lg^\epsilon q)$ words and query time is $T(q) = O(\lg \lg q)$. Plugging these bounds into Lemma 7 gives $n' = \Theta(n / \lg^{1+\epsilon} n)$, and hence $O((\lg^{1+\epsilon} n)(\lg |\Sigma| + \lg \lg n) + j - i)$ retrieval time for suffix pointers sharing $\Omega(\lg^{1+\epsilon} n)$ symbols. Preprocessing time is $O(n \lg |\Sigma|)$ and additional space is $O(n \lg |\Sigma|)$ bits.

The second data structure in [9, 69] has preprocessing time $P(q) = O(q \lg q)$, space $S(q) = O(q)$, and query time $T(q) = O(q^\beta)$ for any fixed value of $0 < \beta < 1$. Consequently, Lemma 7 gives $n' = \Theta(n/\lg n)$ and $O(n^\beta \lg n + j - i) = O(n^\alpha + j - i)$ retrieval time for suffix pointers sharing at least $\Omega(\lg n)$ symbols (by choosing $\alpha > \beta$). Preprocessing time is $O(n \lg |\Sigma|)$ and additional space is $O(n \lg |\Sigma|)$ bits.

4. Text indexing, string searching, and compressed suffix trees. We now describe how to apply our compressed suffix array to obtain a text index, called a *compressed suffix tree*, which is very efficient in time and space complexity. We first show that, despite their extra functionality, compressed suffix trees (and compressed suffix arrays) require the same asymptotic space of $\Theta(n)$ bits as inverted lists in the worst case. Nevertheless, inverted lists are space efficient in practice [72] and can be easily maintained in a dynamic setting.

LEMMA 8. *In the worst case, inverted lists require $\Theta(n)$ bits for a binary text of length n .*

Proof. Let us take a De Bruijn sequence S of length n , in which each substring of $\lg n$ bits is different from the others. Now let the terms in the inverted lists be those obtained by partitioning S into $s = n/k$ disjoint substrings of length $k = 2 \lg n$. Any data structure that implements inverted lists must be able to solve the static dictionary problem on the s terms, and so it requires at least $\lg \binom{2^k}{s} = \Omega(n)$ bits by a simple information-theoretic argument. The upper bound $O(n)$ follows from Theorem 1, and Theorem 4 below, since we can see compressed suffix arrays and suffix trees as generalizations of inverted lists. \square

We now describe our main result on text indexing for constant size alphabets. Here, we are given a pattern string P of m symbols over the alphabet Σ , and we are interested in its occurrences (perhaps overlapping) in a text string T of n symbols (where $\#$ is the n th symbol). We assume that each symbol in Σ is encoded by $\lg |\Sigma|$ bits, which is the case with ASCII and UNICODE text files when two or more symbols are packed in each word.

THEOREM 4. *Given a text string T of length n over an alphabet Σ of constant size, we can build a full text index on T in $O(n \lg |\Sigma|)$ time such that the index occupies $(\epsilon^{-1} + O(1)) n \lg |\Sigma|$ bits, for any fixed value of $0 < \epsilon \leq 1$, and supports the following queries on any pattern string P of m symbols packed into $O(m/\lg_{|\Sigma|} n)$ words:*

(i) *Existential and counting queries can be done in $o(\min\{m \lg |\Sigma|, m + \lg n\})$ time; in particular, they take $O(1)$ time for $m = o(\lg n)$, and $O(m/\lg_{|\Sigma|} n + \lg_{|\Sigma|}^\epsilon n)$ time otherwise.*

(ii) *An enumerative query listing the occ occurrences of P in T can be done in $O(m/\lg_{|\Sigma|} n + occ \lg_{|\Sigma|} n)$ time. We can use auxiliary data structures in $O(n \lg |\Sigma|)$ bits to reduce the search bound to $O(m/\lg_{|\Sigma|} n + occ + (\lg^{1+\epsilon} n)(\lg |\Sigma| + \lg \lg n))$ time, when either $m = \Omega(\lg^{1+\epsilon} n)$ or $occ = \Omega(n^\epsilon)$.*

As a result, an enumerative query can be done in optimal $\Theta(m/\lg_{|\Sigma|} n + occ)$ time for sufficiently large patterns or number of occurrences, namely, when $m = \Omega((\lg^{2+\epsilon} n) \lg_{|\Sigma|} \lg n)$ or $occ = \Omega(n^\epsilon)$.

In order to prove Theorem 4, we first show how to speed up the search on compacted tries in section 4.1. Then we present the index construction in section 4.2. Finally, we give the description of the search algorithm in section 4.3. Let's briefly review three important data structures presented in [45, 59, 62] and that are needed later on.

The first data structure is the Lempel-Ziv (LZ) index [45]. It is a powerful tool

in searching for q -grams (substrings of length q) in T . If we fix $q = \epsilon \lg n$ for any fixed positive constant $\epsilon < 1$, we can build an LZ index on T in $O(n)$ time such that the LZ index occupies $O(n)$ bits and any pattern of length $m \leq \epsilon \lg n$ can be searched in $O(m + occ)$ time. In this special case, we can actually obtain $O(1 + occ)$ time by a suitable table lookup. (Unfortunately, for longer patterns, the LZ index may take $\Omega(n \lg n)$ bits.) The LZ index allows us to concentrate on patterns of length $m > \epsilon \lg n$.

The second data structure is the Patricia trie [59], another powerful tool in text indexing. It is a binary tree that stores a set of distinct binary strings, in which each internal node has two children and each leaf stores a string. For our purposes, we can generalize it to handle alphabets of size $|\Sigma| \geq 2$ by using a $|\Sigma|$ -way tree. Each internal node also keeps an integer (called a skip value) to locate the position of the branching character while descending toward a leaf. Each child arc is implicitly labeled with one symbol of the alphabet. For space efficiency, when there are $t > 2$ child arcs, we can represent the child arcs by a hash table of $O(t)$ entries. In particular, we use a perfect hash function (e.g., see [31, 39]) on keys from Σ , which provides constant lookup time and uses $O(t)$ words of space and $O(t \lg t)$ construction time, in the worst case.

Suffix trees are often implemented by building a Patricia trie on the suffixes of T as follows [35]: First, text T is encoded as a binary sequence of $n \lg |\Sigma|$ bits, and its n suffixes are encoded analogously. Second, a Patricia trie is built upon these suffixes; the resulting suffix tree still has n leaves (not $n \lg |\Sigma|$). Third, searching for P takes $O(m)$ time and retrieves only the suffix pointer in at most two leaves (i.e., the leaf reached by branching with the skip values, and the leaf corresponding to an occurrence). According to our terminology, it requires only $O(1)$ calls to the *lookup* operation in the worst case.

The third data structure is the space-efficient incarnation of binary Patricia tries in [62], which builds upon previous work to succinctly represent binary trees and Patricia tries [16, 42, 60, 61]. When employed to store s out of the n suffixes of T , the regular Patricia trie [59] occupies $O(s \lg n)$ bits. This amount of space usage is the result of three separate factors [15, 16], namely, the Patricia trie topology, the skip values, and the string pointers. Because of our compressed suffix arrays, the string pointers are no longer a problem. For the remaining two items, the space-efficient incarnation of Patricia tries in [62] cleverly avoids the overhead for the Patricia trie topology and the skip values. It is able to represent a Patricia trie storing s suffixes of T with only $O(s)$ bits, *provided that a suffix array is given separately* (which in our case is a compressed suffix array). Searching for query pattern P takes $O(m \lg |\Sigma|)$ time and accesses $O(\min\{m \lg |\Sigma|, s\}) = O(s)$ suffix pointers in the worst case. For each traversed node, its corresponding skip value is computed in time $O(\text{skip_value})$ by accessing the suffix pointers in its leftmost and rightmost descendant leaves. In our terminology, searching requires $O(s)$ calls to *lookup* in the worst case.

4.1. Speeding up Patricia trie search. Before we discuss how to construct the index, we first need to show that search in Patricia tries, which normally proceeds one level at a time, can be improved to sublinear time by processing $\lg n$ bits of the pattern at a time (maybe less if the pattern length is not a multiple of $\lg n$).

Let us first consider the $|\Sigma|$ -way Patricia trie PT outlined in section 4 for storing s binary strings, each of length at least $\lg n$. (For example, they could be some suffixes of the text.) To handle border situations, we assume that these strings are (implicitly) padded with $\lg_{|\Sigma|} n$ symbols $\#$. We will show how to reduce the search time for an m -symbol pattern in PT from $O(m \lg |\Sigma|)$ to $O(m / \lg_{|\Sigma|} n + \lg_{|\Sigma|}^{\epsilon} n)$. Without loss of

generality, it suffices to show how to achieve $O(m/\lg_{|\Sigma|} n + \sqrt{\lg_{|\Sigma|} n})$ time, since this bound extends from $1/2$ to any exponent $\epsilon > 0$. The point is that, in the worst case, we may have to traverse $\Theta(m)$ nodes, so we need a tool to skip most of these nodes. Ideally, we would like to branch downward, matching $\lg n$ bits (or equivalently, $\lg_{|\Sigma|} n$ symbols) in constant time, independently of the number of traversed nodes. For that purpose, we use a perfect hash function h (e.g., see [31]) on keys each of length at most $2 \lg n$ bits. In particular, we use the perfect hash function in [39], which has constant lookup time and takes $O(k)$ words of space and $O(k \lg k)$ construction time on k keys, in the worst case.

First of all, we enumerate the nodes of PT in preorder starting from the root, with number 1. Second, we build hash tables to mimic a downward traversal from a given node i , which is the starting point for searching strings x of length less than or equal to $\lg_{|\Sigma|} n$ symbols. Suppose that, in this traversal, we successfully match all the symbols in x and we reach node j (a descendent of i). In general, there can be further symbols to be added to equal the skip value in j ; let $b \geq 0$ be this number of symbols. We represent the successful traversal in a single entry of the hash table. Namely, we store pair $\langle j, b \rangle$ at position $h(i, x)$, where the two arguments i and x can be seen as a single key of at most $2 \lg n$ bits. Formally, the relation between these parameters must satisfy the following two conditions in the case of a successful search of x from node i :

1. Node j is the node identified by starting out from node i and traversing downward toward the nodes of PT according to the symbols in x ;
2. b is the unique nonnegative integer such that the string corresponding to the path from i to j has prefix x and length $|x| + b$; this condition does not hold for any proper ancestor of j .

The rationale behind conditions 1–2 is that of defining shortcut links from certain nodes i to their descendents j so that each successful branching takes constant time, matches $|x|$ symbols (with b further symbols to check), and skips no more than $|x|$ nodes downward. If the search is unsuccessful, we do not hash any pair.

The key mechanism that makes the above scheme efficient is that we adaptively follow the trie topology of Patricia so that the strings that we hash are not all possible substrings of $\lg_{|\Sigma|} n$ (or $\sqrt{\lg_{|\Sigma|} n}$) symbols, but only a subset of those that start at the distinct nodes in the Patricia trie. Using an uncompact trie would make this method inefficient. To see why, let us examine a Patricia edge corresponding to a substring of length l . We hash only its first $\lg_{|\Sigma|} n$ (or $\sqrt{\lg_{|\Sigma|} n}$) symbols because the rest of the symbols are uniquely identified (and we can skip them). Using an uncompact trie would force us to traverse further $b = l - \lg_{|\Sigma|} n$ (or $b = l - \sqrt{\lg_{|\Sigma|} n}$) nodes.

In order to keep small the number of shortcut links, we set up two hash tables H_1 and H_2 . The first table stores entries

$$H_1[h(i, x)] = \langle j, b \rangle$$

such that all strings x consist of $|x| = \lg_{|\Sigma|} n$ symbols, and the shortcut links stored in H_1 are selected adaptively by a top-down traversal of PT . Namely, we create all possible shortcut links from the root. This step links the root to a set of descendents. We recursively link each of these nodes to its descendents in the same fashion. Note that PT is partitioned into subtrees of depth at most $\lg_{|\Sigma|} n$.

We set up the second table H_2 analogously. We examine each individual subtree and start from the root of the subtree by using strings of length $|x| = \sqrt{\lg_{|\Sigma|} n}$ symbols. Note that the total number of entries in H_1 and H_2 is bounded by the number of nodes in PT , namely, $O(s)$.

In summary, the preprocessing consists of a double traversal of PT followed by the construction of H_1 and H_2 , in $O(s \lg s + n)$ worst-case time and $O(s)$ words of space. In the general case, we go on recursively and build ϵ^{-1} hash tables, whose total number of entries is still $O(s)$. The preprocessing time does not change asymptotically.

We are now ready to describe the search of a pattern (encoded in binary) in the Patricia trie PT thus augmented. It suffices to show how to match its longest prefix. We compute hash function $h(i, x)$ with i being the root of PT and x being the concatenation of the first $\lg_{|\Sigma|} n$ symbols in the pattern. Then we branch quickly from the root by using $H_1[h(i, x)]$. If the hash lookup in H_1 succeeds and gives pair $\langle j, b \rangle$, we skip the next b symbols in the pattern and recursively search in node j with the next $\lg_{|\Sigma|} n$ symbols in the pattern (read in $O(1)$ time). Instead, if the hash lookup fails (i.e., no pair is found or fewer than $\lg_{|\Sigma|} n$ symbols are left in the pattern), we switch to H_2 and take only the next $\sqrt{\lg_{|\Sigma|} n}$ symbols in the pattern to branch further in PT . Here the scheme is the same as that of H_1 , except that we compare $\sqrt{\lg_{|\Sigma|} n}$ symbols at a time. Finally, when we fail branching again, we have to match no more than $\sqrt{\lg_{|\Sigma|} n}$ symbols remaining in the pattern. We complete this task by branching in the standard way, one symbol a time. The rest of the search is identical to the standard procedure of Patricia tries. This completes the description of the search in PT .

LEMMA 9. *Given a Patricia trie PT storing s strings of at least $\lg_{|\Sigma|} n$ symbols each over the alphabet Σ , we can preprocess PT in $O(s \lg s + n)$ time so that searching a pattern of length m requires $O(m/\lg_{|\Sigma|} n + \lg_{|\Sigma|}^\epsilon n)$ time.*

Note that a better search bound in Lemma 9 does not improve the final search time obtained in Theorem 4.

Finally, let us consider a space-efficient Patricia trie [62]. The speedup we need while searching is easier to obtain. We need not skip nodes, but need only compare $\Theta(\lg n)$ bits at a time in constant time by precomputing a suitable table. The search cost is therefore $O(m/\lg_{|\Sigma|} n)$ plus a linear cost proportional to the number of traversed nodes.

A general property of our speedup of Patricia tries is that we do not increase the original number of *lookup* calls originating from the data structures.

4.2. Index construction. We blend the tools mentioned so far with our compressed suffix arrays of section 3 to design a hybrid index data structure, called the *compressed suffix tree*, which follows the multilevel scheme adopted in [17, 62]. Because of the LZ index, it suffices to describe how to support searching of patterns of length $m > \epsilon \lg n$. We assume that $0 < \epsilon \leq 1/2$, as the case $1/2 < \epsilon \leq 1$ requires minor modifications.

Given text T in input, we build its suffix array SA in a temporary area, in $O(n \lg |\Sigma|)$ time via the suffix tree of T . At this point, we start building the $O(\epsilon^{-1})$ levels of the compressed suffix tree in top-down order, after which we remove SA as follows:

1. At the first level, we build a regular Patricia trie PT^1 augmented with the shortcut links as mentioned in Lemma 9. The leaves of PT^1 store the $s_1 = n/\lg_{|\Sigma|} n$ suffixes pointed to by $SA[1]$, $SA[1 + \lg_{|\Sigma|} n]$, $SA[1 + 2\lg_{|\Sigma|} n]$, \dots . This implicitly splits SA into s_1 subarrays of size $\lg_{|\Sigma|} n$, except the last one (which can be smaller).

Complexity. The size of PT^1 is $O(s_1 \lg n) = O(n \lg |\Sigma|)$ bits. It can be built in $O(n \lg |\Sigma|)$ time by a variation of the standard suffix tree construction [51, 52] and the preprocessing described in Lemma 9.

2. At the second level, we process the s_1 subarrays from the first level, and create

s_1 space-efficient Patricia tries [62], denoted $PT_1^2, PT_2^2, \dots, PT_{s_1}^2$. We associate the i th Patricia PT_i^2 with the i th subarray. Assume without loss of generality that the subarray consists of $SA[h+1], SA[h+2], \dots, SA[h+\lg_{|\Sigma|} n]$ for a value of $0 \leq h \leq n - \lg_{|\Sigma|} n$. We build PT_i^2 upon the $s_2 = \lg_{|\Sigma|}^{\epsilon/2} n$ suffixes pointed to by $SA[h+1], SA[h+1+\lg_{|\Sigma|}^{1-\epsilon/2} n], SA[h+1+2\lg_{|\Sigma|}^{1-\epsilon/2} n], \dots$. This process splits each subarray into smaller subarrays, where each subarray is of size $\lg_{|\Sigma|}^{1-\epsilon/2} n$.

Complexity. The size of each PT_i^2 is $O(s_2)$ bits without accounting for the suffix array, and its construction takes $O(s_2)$ time [62]. Hence, the total size is $O(s_1 s_2) = O(n/\lg_{|\Sigma|}^{1-\epsilon} n)$ bits and the total processing time is $O(n \lg |\Sigma|)$.

3. In the remaining $2\epsilon^{-1} - 2$ intermediate levels, we proceed as in the second level. Each new level splits every subarray into $s_2 = \lg_{|\Sigma|}^{\epsilon/2} n$ smaller subarrays and creates a set of space-efficient Patricia tries of size $O(s_2)$ each. We stop when we are left with small subarrays of size at most s_2 . We build space-efficient Patricia tries on all the remaining entries of these small subarrays.

Complexity. For each new level thus created, the total size is $O(n/\lg_{|\Sigma|}^{\epsilon} n)$ bits and the total processing time is $O(n \lg |\Sigma|)$.

4. At the last level, we execute *compress* on the suffix array SA , store its compressed version in the level, and delete SA from the temporary area.

Complexity. By Theorem 2, the total size is $(\epsilon^{-1} + O(1))n \lg |\Sigma|$ bits; accessing a pointer through a call to *lookup* takes $O(\lg_{|\Sigma|}^{\epsilon/2} n)$ time; the cost of *compress* is $O(n \lg |\Sigma|)$ time. (Note that we can fix the value of ϵ arbitrarily when executing *compress*.)

By summing over the levels, we obtain that the compressed suffix tree of T takes $O(n \lg |\Sigma|)$ bits and $O(n \lg |\Sigma|)$ construction time. Temporary storage is $O(n \lg n)$ bits.

4.3. Search algorithm. We now have to show that searching for an arbitrary pattern P in the text T costs $O(m/\lg_{|\Sigma|} n + \lg_{|\Sigma|}^{\epsilon} n)$ time. The search locates the leftmost occurrence and the rightmost occurrence of P as a prefix of the suffixes represented in SA , without having SA stored explicitly. Consequently, a successful search determines two positions $i \leq j$ such that the sequence $i, i+1, \dots, j$ is maximal (according to the definition given before Theorem 3) and $SA[i], SA[i+1], \dots, SA[j]$ contain the pointers to the suffixes that begin with P . The counting query returns $j-i+1$, and the existence checks whether there are any matches at all. The enumerative query executes the $j-i+1$ queries $lookup(i), lookup(i+1), \dots, lookup(j)$ to list all the occurrences.

We restrict our discussion to finding the leftmost occurrence of P ; finding the rightmost is analogous. We search at each level of the compressed suffix tree in section 4.2. We examine the levels in a top-down manner. While searching in the levels, we execute $lookup(i)$ whenever we need the i th pointer of the compressed SA . We begin by searching P at the first level. We perform the search on PT^1 in the bounds stated in Lemma 9. As a result of the first search, we locate a subarray at the second level, say, the i_1 th subarray. We go on and search in $PT_{i_1}^2$ according to the method for space-efficient Patricia tries described at the end of section 4.1. We repeat the latter search for all the intermediate levels. We eventually identify a position at the last level, namely, the level which contains the compressed suffix array. This position corresponds to the leftmost occurrence of P in SA .

The complexity of the search procedure is $O(m/\lg_{|\Sigma|} n + \lg_{|\Sigma|}^{\epsilon} n)$ time at the first level by Lemma 9. The intermediate levels cost $O(m/\lg_{|\Sigma|} n + s_2)$ time each, giving a total of $O(m/\lg_{|\Sigma|} n + \lg_{|\Sigma|}^{\epsilon} n)$. We have to account for the cost of the *lookup*

operations. These calls originated from the several levels. In the first level, we call *lookup* $O(1)$ times; in the $2\epsilon^{-1} - 1$ intermediate levels we call *lookup* $O(s_2)$ times each. Multiplying these calls by the $O(\lg_{|\Sigma|}^{\epsilon/2} n)$ cost of *lookup* as given in Theorem 1 (using $\epsilon/2$ in place of ϵ), we obtain $O(\lg_{|\Sigma|}^{\epsilon} n)$ time in addition to $O(m/\lg_{|\Sigma|} n + \lg_{|\Sigma|}^{\epsilon} n)$. Finally, the cost of retrieving all the occurrences is the one stated in Theorem 3, whose hypothesis is satisfied because the suffixes pointed to by $SA[i]$ and $SA[j]$ are, respectively, the leftmost and rightmost sharing $m = \Omega(\lg n)$ symbols. Combining this cost with the $O(\lg_{|\Sigma|}^{\epsilon} n)$ cost for retrieving any single pointer in Theorem 1, we obtain $O(m/\lg_{|\Sigma|} n + occ \lg_{|\Sigma|}^{\epsilon} n)$ time when both conditions $m \in [\epsilon \lg n, o(\lg^{1+\epsilon} n)]$ and $occ = o(n^{\epsilon})$ hold, and in $O(m/\lg_{|\Sigma|} n + occ + (\lg^{1+\epsilon} n)(\lg |\Sigma| + \lg \lg n))$ time otherwise. This argument completes the proof of Theorem 4 on the complexity of our text index.

5. Conclusions. We have presented the first indexing data structure for a text T of n symbols over alphabet Σ that achieves, in the worst case, efficient lookup time and linear space. For many scenarios, the space requirement is actually sublinear in practice. Specifically, our algorithm uses $o(\min\{m \lg |\Sigma|, m + \lg n\})$ search time and $(\epsilon^{-1} + O(1)) n \lg |\Sigma|$ bits of space (where T requires $n \lg |\Sigma|$ bits). Our method is based upon notions of compressed suffix arrays and suffix trees. Given any pattern P of m symbols encoded in $m \lg |\Sigma|$ bits, we can count the number of occurrences of P in T in $o(\min\{m \lg |\Sigma|, m + \lg n\})$ time. Namely, searching takes $O(1)$ time when $m = o(\lg n)$, and $O(m/\lg_{|\Sigma|} n + \lg_{|\Sigma|}^{\epsilon} n)$ time otherwise. We achieve optimal $O(m/\lg_{|\Sigma|} n)$ search time for sufficiently large $m = \Omega(\lg_{|\Sigma|}^{1+\epsilon} n)$. For an enumerative query retrieving all occ occurrences with sufficiently long patterns, namely, $m = \Omega((\lg^{2+\epsilon} n) \lg_{|\Sigma|} \lg n)$, we obtain a total search bound of $O(m/\lg_{|\Sigma|} n + occ)$, which is optimal. Namely, searching takes $O(m/\lg_{|\Sigma|} n + occ \lg_{|\Sigma|}^{\epsilon} n)$ time when both conditions $m \in [\epsilon \lg n, o(\lg^{1+\epsilon} n)]$ and $occ = o(n^{\epsilon})$ hold, and $O(m/\lg_{|\Sigma|} n + occ + (\lg^{1+\epsilon} n)(\lg |\Sigma| + \lg \lg n))$ time otherwise. Crucial to our results are functions Ψ_k and Φ_k (see section 2), which are the building blocks of many other results in compressed text indexing.

An interesting open problem is to improve upon our $O(n)$ -bit compressed suffix array so that each call to *lookup* takes constant time. Such an improvement would decrease the output-sensitive time of the enumerative queries to $O(occ)$ also when $m \in [\epsilon \lg n, o(\lg^{1+\epsilon} n)]$ and $occ = o(n^{\epsilon})$. Another possibility for that is to devise a range query data structure that improves the data structures at the end of section 3.4. This, in turn, would improve Theorems 3 and 4. A related question is to characterize combinatorially the permutations that correspond to suffix arrays. A better understanding of the correspondence may lead to more efficient compression methods. Additional open problems are listed in [62]. The kinds of queries examined in this paper are very basic and involve exact occurrences of the pattern strings. They are often used as preliminary filters so that more sophisticated queries can be performed on a smaller amount of text. An interesting extension would be to support some sophisticated queries directly, such as those that tolerate a small number of errors in the pattern match [4, 12, 35, 70].

REFERENCES

- [1] S. ALSTRUP, G. S. BRODAL, AND T. RAUHE, *New data structures for orthogonal range searching*, in Proceedings of the 41st Annual IEEE Symposium on Foundations of Computer Science, 2000, pp. 198–207.

- [2] A. AMIR AND G. BENSON, *Efficient two-dimensional compressed matching*, in Proceedings of the IEEE Data Compression Conference, J. A. Storer and M. Cohn, eds., IEEE Computer Society Press, Los Alamitos, CA, 1992, pp. 279–288.
- [3] A. AMIR, G. BENSON, AND M. FARACH, *Let sleeping files LIE: Pattern matching in Z-compressed files*, J. Comput. System Sci., 52 (1996), pp. 299–307.
- [4] A. AMIR, D. KESELMAN, G. M. LANDAU, M. LEWENSTEIN, N. LEWENSTEIN, AND M. RODEH, *Text indexing and dictionary matching with one error*, J. Algorithms, 37 (2000), pp. 309–325.
- [5] A. ANDERSSON, N. J. LARSSON, AND K. SWANSON, *Suffix trees on words*, Algorithmica, 23 (1999), pp. 246–260.
- [6] A. ANDERSSON AND S. NILSSON, *Efficient implementation of suffix trees*, Software Practice and Experience, 25 (1995), pp. 129–141.
- [7] A. APOSTOLICO, *The myriad virtues of suffix trees*, in Combinatorial Algorithms on Words, A. Apostolico and Z. Galil, eds., NATO Advanced Science Institutes Series F: Computer and System Sciences 12, Springer-Verlag, Berlin, 1985, pp. 85–96.
- [8] A. APOSTOLICO, C. ILIOPOULOS, G. M. LANDAU, B. SCHIEBER, AND U. VISHKIN, *Parallel construction of a suffix tree with applications.*, Algorithmica, 3 (1988), pp. 347–365.
- [9] J. L. BENTLEY AND H. A. MAURER, *Efficient worst-case data structures for range searching*, Acta Inform., 13 (1980), pp. 155–168.
- [10] A. BLUMER, J. BLUMER, D. HAUSSLER, A. EHRENFUCHT, M. T. CHEN, AND J. SEIFERAS, *The smallest automation recognizing the subwords of a text*, Theoret. Comput. Sci., 40 (1985), pp. 31–55.
- [11] A. BLUMER, J. BLUMER, D. HAUSSLER, R. MCCONNELL, AND A. EHRENFUCHT, *Complete inverted files for efficient text retrieval and analysis*, J. ACM, 34 (1987), pp. 578–595.
- [12] G. S. BRODAL AND L. GAŚSIENIEC, *Approximate dictionary queries*, in Proceedings of the 7th Annual Symposium on Combinatorial Pattern Matching, D. S. Hirschberg and E. W. Myers, eds., Lecture Notes in Comput. Sci. 1075, Springer-Verlag, Berlin, New York, 1996, pp. 65–74.
- [13] A. BRODNIK AND J. I. MUNRO, *Membership in constant time and almost-minimum space*, SIAM J. Comput., 28 (1999), pp. 1627–1640.
- [14] M. BURROWS AND D. J. WHEELER, *A Block Sorting Data Compression Algorithm*, Tech. report, Digital Systems Research Center, Palo Alto, CA, 1994.
- [15] D. CLARK, *Compact Pat Trees*, Ph.D. thesis, Department of Computer Science, University of Waterloo, Waterloo, Ontario, Canada, 1996.
- [16] D. R. CLARK AND J. I. MUNRO, *Efficient suffix trees on secondary storage (extended abstract)*, in Proceedings of the Seventh Annual ACM-SIAM Symposium on Discrete Algorithms, SIAM, Philadelphia, 1996, pp. 383–391.
- [17] L. COLUSSI AND A. DE COL, *A time and space efficient data structure for string searching on large texts*, Inform. Process. Lett., 58 (1996), pp. 217–222.
- [18] M. CROCHEMORE, *Transducers and repetitions*, Theoret. Comput. Sci., 45 (1986), pp. 63–86.
- [19] M. CROCHEMORE AND D. PERRIN, *Two-way string matching*, J. ACM, 38 (1991), pp. 651–675.
- [20] M. CROCHEMORE AND W. RYTTER, *Text Algorithms*, Oxford University Press, Oxford, UK, 1994.
- [21] E. S. DE MOURA, G. NAVARRO, AND N. ZIVIANI, *Indexing compressed text*, in Proceedings of the South American Workshop on String Processing, Carleton University Press, Ottawa, Ontario, Canada, 1997, pp. 95–111.
- [22] E. D. DEMAINE AND A. LÓPEZ-ORTIZ, *A linear lower bound on index size for text retrieval*, in Proceedings of the 12th Annual ACM-SIAM Symposium on Discrete Algorithms, SIAM, Philadelphia, 2001, pp. 289–294.
- [23] P. ELIAS, *Efficient storage and retrieval by content and address of static files*, J. ACM, 21 (1974), pp. 246–260.
- [24] M. FARACH AND M. THORUP, *String matching in Lempel-Ziv compressed strings*, Algorithmica, 20 (1998), pp. 388–404.
- [25] M. FARACH-COLTON, *Optimal suffix tree construction with large alphabets*, in Proceedings of the 38th Annual IEEE Symposium on Foundations of Computer Science, Miami Beach, FL, 1997, pp. 137–143.
- [26] M. FARACH-COLTON, P. FERRAGINA, AND S. MUTHUKRISHNAN, *On the sorting-complexity of suffix tree construction*, J. ACM, 47 (2000), pp. 987–1011.
- [27] M. FARACH-COLTON AND S. MUTHUKRISHNAN, *Optimal logarithmic time randomized suffix tree construction*, in Automata, Languages, and Programming, 23rd International Colloquium, F. Meyer auf der Heide and B. Monien, eds., Lecture Notes in Comput. Sci. 1099, Springer-Verlag, Berlin, New York, 1996, pp. 550–561.

- [28] P. FERRAGINA AND R. GROSSI, *The String B-tree: a new data structure for string search in external memory and its applications*, J. ACM, 46 (1999), pp. 236–280.
- [29] P. FERRAGINA AND G. MANZINI, *Opportunistic data structures with applications*, in Proceedings of the 41st Annual IEEE Symposium on Foundations of Computer Science, 2000, pp. 390–398.
- [30] P. FERRAGINA AND G. MANZINI, *An experimental study of an opportunistic index*, in Proceedings of the 12th Annual ACM-SIAM Symposium on Discrete Algorithms, SIAM, Philadelphia, 2001, pp. 269–278.
- [31] M. L. FREDMAN, J. KOMLÓS, AND E. SZEMERÉDI, *Storing a sparse table with $O(1)$ worst case access time*, J. ACM, 31 (1984), pp. 538–544.
- [32] A. GÁL AND P. B. MILTENSEN, *Automata, Languages and Programming*, in Proceedings of the 30th International Colloquium, Lecture Notes in Comput. Sci. 2719, Springer, Berlin, New York, 2003, pp. 332–344.
- [33] Z. GALIL AND J. SEIFERAS, *Time-space-optimal string matching*, J. Comput. System Sci., 26 (1983), pp. 280–294.
- [34] R. GIEGERICH, S. KURTZ, AND J. STOYE, *Efficient implementation of lazy suffix trees*, in Proceedings of the 3rd Workshop on Algorithm Engineering, J. S. Vitter and C. D. Zaroliagis, eds., Lecture Notes in Comput. Sci. 1668, Springer-Verlag, Berlin, 1999, pp. 30–42.
- [35] G. H. GONNET, R. A. BAEZA-YATES, AND T. SNIDER, *New indices for text: PAT trees and PAT arrays*, in Information Retrieval: Data Structures And Algorithms, Prentice-Hall, Englewood Cliffs, NJ, 1992, pp. 66–82.
- [36] R. GROSSI, A. GUPTA, AND J. S. VITTER, *High-order entropy-compressed text indexes*, in Proceedings of the 14th Annual ACM-SIAM Symposium on Discrete Algorithms, SIAM, Philadelphia, 2003, pp. 841–850.
- [37] R. GROSSI AND J. S. VITTER, *Compressed suffix arrays and suffix trees with applications to text indexing and string matching (extended abstract)*, in Proceedings of the 32nd Annual ACM Symposium on the Theory of Computing, Portland, OR, 2000, pp. 397–406.
- [38] D. GUSFIELD, *Algorithms on Strings, Trees and Sequences: Computer Science and Computational Biology*, Cambridge University Press, Cambridge, UK, 1997.
- [39] T. HAGERUP, P. B. MILTENSEN, AND R. PAGH, *Deterministic dictionaries*, J. Algorithms, 41 (2001), pp. 69–85.
- [40] W.-K. HON, K. SADAKANE, AND W.-K. SUNG, *Breaking a time-and-space barrier in constructing full-text indices*, in Proceedings of the 44th Annual IEEE Symposium on Foundations of Computer Science (FOCS), 2003, pp. 251–260.
- [41] R. W. IRVING, *Suffix Binary Search Trees*, Tech. report TR-1995-7, Computing Science Department, University of Glasgow, Glasgow, UK, 1995.
- [42] G. JACOBSON, *Space-efficient static trees and graphs*, in Proceedings of the 30th Annual IEEE Symposium on Foundations of Computer Science, 1989, pp. 549–554.
- [43] G. JACOBSON, *Succinct Static Data Structures*, Tech. report CMU-CS-89-112, Department of Computer Science, Carnegie-Mellon University, Pittsburgh, PA, 1989.
- [44] J. KÄRKKÄINEN, *Suffix cactus: A cross between suffix tree and suffix array*, in Combinatorial Pattern Matching, Lecture Notes in Comput. Sci. 937, Springer, Berlin, New York, 1995, pp. 191–204.
- [45] J. KÄRKKÄINEN AND E. SUTINEN, *Lempel-Ziv index for q -grams*, Algorithmica, 21 (1998), pp. 137–154.
- [46] J. KÄRKKÄINEN AND E. UKKONEN, *Lempel-Ziv parsing and sublinear-size index structures for string matching*, in Proceedings of the 3rd South American Workshop on String Processing, N. Ziviani, R. Baeza-Yates, and K. Guimarães, eds., Carleton University Press, Ottawa, Ontario, Canada, 1996, pp. 141–155.
- [47] J. KÄRKKÄINEN AND E. UKKONEN, *Sparse suffix trees*, in Computing and Combinatorics, Lecture Notes in Comput. Sci. 1090, Springer, Berlin, 1996, pp. 219–230.
- [48] D. E. KNUTH, *Sorting and Searching*, 2nd ed., The Art of Computer Programming 3, Addison-Wesley, Reading, MA, 1998.
- [49] D. E. KNUTH, J. H. MORRIS, JR., AND V. R. PRATT, *Fast pattern matching in strings*, SIAM J. Comput., 6 (1977), pp. 323–350.
- [50] S. R. KOSARAJU, *Real-time pattern matching and quasi-real-time construction of suffix tree*, in Proceedings of the 26th ACM Symposium on the Theory of Computing, ACM, New York, pp. 310–316.
- [51] S. R. KOSARAJU AND A. L. DELCHER, *Large-scale assembly of DNA strings and space-efficient construction of suffix trees*, in Proceedings of the Twenty-Seventh Annual ACM Symposium on the Theory of Computing, Las Vegas, NV, 1995, pp. 169–177.

- [52] S. R. KOSARAJU AND A. L. DELCHER, *Correction: Large-scale assembly of DNA strings and space-efficient construction of suffix trees*, in Proceedings of the 28th Annual ACM Symposium on the Theory of Computing, Philadelphia, PA, 1996, p. 659.
- [53] S. KURTZ, *Reducing the Space Requirement of Suffix Trees*, Software Practice and Experience, 29 (1999), pp. 1149–1171.
- [54] V. MÄKINEN, *Compact suffix array*, in Combinatorial Pattern Matching, Lecture Notes in Comput. Sci. 1848, Springer, Berlin, New York, 2000, pp. 305–319.
- [55] U. MANBER AND G. MYERS, *Suffix arrays: A new method for on-line string searches*, SIAM J. Comput., 22 (1993), pp. 935–948.
- [56] U. MANBER AND S. WU, *GLIMPSE: A tool to search through entire file systems*, in Proceedings of the USENIX Winter 1994 Technical Conference, 1994, pp. 23–32.
- [57] E. M. MCCREIGHT, *A space-economical suffix tree construction algorithm*, J. ACM, 23 (1976), pp. 262–272.
- [58] A. MOFFAT AND J. ZOBEL, *Self-indexing inverted files for fast text retrieval*, ACM Trans. Inform. Systems, 14 (1996), pp. 349–379.
- [59] D. R. MORRISON, *PATRICIA—Practical algorithm to retrieve information coded In alphanumeric*, J. ACM, 15 (1968), pp. 514–534.
- [60] J. I. MUNRO, *Tables*, in Foundations of Software Technology and Theoretical Computer Science, Lecture Notes in Comput. Sci. 1180, Springer, Berlin, pp. 37–42.
- [61] J. I. MUNRO AND V. RAMAN, *Succinct representation of balanced parentheses, static trees and planar graphs*, in Proceedings of the 38th Annual IEEE Symposium on Foundations of Computer Science, 1997, pp. 118–126.
- [62] J. I. MUNRO, V. RAMAN, AND S. S. RAO, *Space efficient suffix trees*, J. Algorithms, 39 (2001), pp. 205–222.
- [63] R. PAGH, *Low redundancy in static dictionaries with constant query time*, SIAM J. Comput., 31 (2001), pp. 353–363.
- [64] K. SADAKANE, *Compressed text databases with efficient query algorithms based on the compressed suffix array*, in Proceedings of ISAAC '00, Lecture Notes in Comput. Sci. 1969, Springer, Berlin, New York, 2000, pp. 410–421.
- [65] K. SADAKANE, *Succinct representations of lcp information and improvements in the compressed suffix arrays*, in Proceedings of the 13th Annual ACM-SIAM Symposium on Discrete Algorithms, SIAM, Philadelphia, 2002, pp. 225–232.
- [66] S. C. SAHINALP AND U. VISHKIN, *Symmetry breaking for suffix tree construction*, in Proceedings of the 26th Annual Symposium on the Theory of Computing, ACM, 1994, pp. 300–309.
- [67] E. UKKONEN, *On-line construction of suffix trees*, Algorithmica, 14 (1995), pp. 249–260.
- [68] P. WEINER, *Linear pattern matching algorithm*, in Proceedings of the 14th Annual IEEE Symposium on Switching and Automata Theory, 1973, pp. 1–11.
- [69] D. E. WILLARD, *On the application of sheared retrieval to orthogonal range queries*, in Proceedings of the Second Annual Symposium on Computational Geometry, ACM, New York, 1986, pp. 80–89.
- [70] A. C. YAO AND F. F. YAO, *Dictionary look-up with one error*, J. Algorithms, 25 (1997), pp. 194–202.
- [71] N. ZIVIANI, E. S. DE MOURA, G. NAVARRO, AND R. BAEZA-YATES, *Compression: A key for next-generation text retrieval systems*, IEEE Comput., 33 (2000), pp. 37–44.
- [72] J. ZOBEL, A. MOFFAT, AND K. RAMAMOCHANARAO, *Inverted files versus signature files for text indexing*, ACM Trans. Database Systems, 23 (1998), pp. 453–490.

RECOGNIZING MORE UNSATISFIABLE RANDOM k -SAT INSTANCES EFFICIENTLY*

JOEL FRIEDMAN[†], ANDREAS GOERDT[‡], AND MICHAEL KRIVELEVICH[§]

Abstract. It is known that random k -SAT instances with at least cn clauses, where $c = c_k$ is a suitable constant, are unsatisfiable (with high probability). We consider the problem to certify efficiently the unsatisfiability of such formulas. A backtracking-based algorithm of Beame et al. [*SIAM J. Comput.*, 31 (2002), pp. 1048–1075] shows that k -SAT instances with at least $n^{k-1}/(\log n)^{k-2}$ clauses can be certified unsatisfiable in polynomial time. We employ spectral methods to improve on this bound. For even $k \geq 4$ we present a polynomial time algorithm which certifies random k -SAT instances with at least $n^{(k/2)+o(1)}$ clauses as unsatisfiable (with high probability). For odd k we focus on 3-SAT instances and obtain an efficient algorithm for formulas with at least $n^{3/2+\varepsilon}$ clauses, where $\varepsilon > 0$ is an arbitrary constant.

Key words. random structures, random satisfiability, spectral methods

AMS subject classifications. 05C80, 68R05, 34L15, 34L20, 03B05

DOI. 10.1137/S009753970444096X

Introduction. We study the complexity of certifying unsatisfiability of random k -SAT instances (or k -CNF formulas) over n propositional variables. Our discussion refers to k fixed and then letting n be sufficiently large. The probability space of random k -SAT instances has been widely studied in recent years for several good reasons. A somewhat arbitrary selection of the most recent literature is [Ac2000], [Fr99], [Be et al2002], [AcSo2000], [AcMo2002], [AcPe2004].

One of the reasons for studying random k -SAT instances is that they have the following sharp threshold behavior [Fr99]: there exists a function $c = c_k(n)$ such that for any $\varepsilon > 0$, formulas with at most $(1 - \varepsilon) \cdot c \cdot n$ clauses are satisfiable, whereas formulas with at least $(1 + \varepsilon) \cdot c \cdot n$ are unsatisfiable with high probability (that means with probability tending to 1 when n goes to infinity). In fact, it is not known if $c_k(n)$ can be taken to be a constant (i.e., if the limit of $c_k(n)$ as $n \rightarrow \infty$ exists). It might be that $c_k = c_k(n)$ satisfying the aforementioned threshold property depends on n . However, it is known that c_k is at most $2^k \cdot \ln 2$, and the general conjecture is that c_k converges to a constant. For formulas with at least $2^k \cdot (\ln 2) \cdot n$ clauses the expected number of satisfying assignments of a random formula tends to 0 and the formulas are unsatisfiable with high probability. As it is well known that the satisfiability problem for random 2-SAT instances is solvable in polynomial time, the most interesting case is that of 3-SAT. Accordingly, much work is spent to approximate the value of c_3 . Currently, the best results are that c_3 is at least 3.26 [AcSo2000] improved to the recent 3.52 [KaKiLa2002] and at most 4.601 [KiKrKr98]

*Received by the editors February 12, 2004; accepted for publication (in revised form) May 9, 2005; published electronically October 17, 2005.

<http://www.siam.org/journals/sicomp/35-2/44096.html>

[†]Department of Mathematics, University of British Columbia, Vancouver, BC V6T 1Z2, Canada (jf@math.ubc.ca, www.math.ubc.ca/~jf).

[‡]Fakultät für Informatik, TU Chemnitz, 09107 Chemnitz, Germany (goerd@informatik.tu-chemnitz.de, www.tu-chemnitz.de/informatik/HomePages/TI). The work of this author was partially supported by the DFG.

[§]Department of Mathematics, Faculty of Exact Sciences, Tel Aviv University, Tel Aviv 69978, Israel (krivelev@math.tau.ac.il). The work of this author was partially supported by a USA-Israeli BSF grant and by a grant from the Israel Science Foundation.

improved to 4.506 [DuBoMa2000]. For $k = 2$ the threshold is known; we have $c_2 = 1$ [ChRe92], [Go96].

The algorithmic interest in this threshold is due to the empirical observation that random k -SAT instances at the threshold, i.e., with around $c_k n$ random clauses, are seemingly hard instances. The following behavior has been reported consistently in experimental studies with suitably optimized backtracking algorithms searching for a satisfying assignment (see, for example, [SeMiLe96], [CrAu96]): the average running time is quite low for instances below the threshold. For 3-SAT instances we observe that almost all formulas with at most $4n$ clauses are satisfiable and it is quite easy to find a satisfying assignment. A precipitous increase in the average running time is observed at the threshold. For 3-SAT, about half of the formulas with $4.2n$ clauses are satisfiable and it is difficult to decide whether a formula is satisfiable or not. Finally, a speedy decline to lower complexity is observed beyond the threshold. For 3-SAT, almost all formulas with $4.5n$ clauses are unsatisfiable and the running time decreases again (in spite of the fact that now the whole backtracking tree must always be searched.)

There are no general complexity theoretic results relating the threshold to hardness. The following observation is trivial: if we can efficiently certify almost all instances with dn clauses, where d above the threshold unsatisfiable, then we can certify almost all instances with $d'n$ clauses, $d' > d$, as unsatisfiable by simply chopping off the superfluous clauses. An analogous fact holds below the threshold, where we extend a given formula with some random clauses. Of course, similar remarks apply to the size of clauses.

The relationship between hardness and thresholds is not restricted to satisfiability. It has also been observed for k -colorability of random graphs with a linear number of edges. In [PeWe89] a peak in running time seemingly related to the threshold is reported. The existence of a threshold is proved in [AcFr99], but again the value and convergence to a constant are known only experimentally. For the subset sum problem, which is of a quite different nature, the threshold is known. Some discussion related to hardness is found in [ImNa96]. For the similarly looking number partitioning problem threshold results can be found in [BoChPi2001].

Abandoning the general complexity theoretic point of view and looking at concrete algorithms, we see that the following results are known for random k -SAT instances: all progress approximating the threshold from below is based on the analysis of rather simple polynomial time heuristics and is mostly restricted to clause size $k = 3$. In fact, the most advanced heuristic analyzed [AcSo2000] finds only a satisfying assignment with probability of at least ε , where $\varepsilon > 0$ is a small constant, for 3-SAT formulas with at most $3.26n$ clauses. The same applies to the recent improvement to $3.52n$ clauses in [KaKiLa2002]. The heuristic in [FrSu96] finds a satisfying assignment almost always for random 3-SAT instances with at most $3.003n$ clauses. On the other hand the progress made in approximating the threshold from above does not provide us at all with efficient algorithms certifying the unsatisfiability of the formula at hand. Only the expectation of the number of satisfying assignments is calculated and is shown to tend to 0.

In fact, beyond the threshold we have negative results: for arbitrary but fixed $d \geq 2^k \cdot \ln 2$, random k -SAT instances with dn clauses (are unsatisfiable and) have only resolution proofs with an exponential number, that is, with at least $2^{\Omega(n)}$ clauses with high probability [ChSz88]. This has been improved upon by [Fu95], [BePi96], and [Be at al2002], all of whom proved (exponential) lower bounds for somewhat larger clause/variable ratios. Note that a lower bound on the size of the resolution proofs

provides a lower bound on the number of nodes in *any* classical backtracking tree as generated by any variant of the well-known Davis–Putnam procedure.

Provably polynomial time results beyond the threshold are rather limited at present: in [Fu95] it is shown that random k -SAT formulas with at least n^{k-1} clauses allow for polynomial size resolution proofs with high probability. This is strengthened in [Be et al2002] to the best result known at present: for at least $n^{k-1}/(\log n)^{k-2}$ random clauses a backtracking-based algorithm proves unsatisfiability in polynomial time with high probability. (The result of Beame et al. is slightly stronger, as it applies to formulas with $\Omega(n^{k-1}/(\log n)^{k-2})$ random clauses.)

We extend the region, where a provably polynomial time algorithm exists. For even $k \geq 4$ we give an algorithm which works when the number of clauses is only n to a constant fraction of k (with high probability), that is, for formulas with at least $\text{POLY}(\log n) \cdot n^{k/2} = n^{(k/2)+o(1)}$ clauses, where POLY denotes a polynomial growing sufficiently fast (a degree of 7 is sufficient). A preliminary version of this result has been published in [GoKr 2001]. To obtain our result we leave the area of strictly combinatorial algorithms considered up to this point. Instead we associate a graph with a given formula and show how to certify unsatisfiability of the formula with the help of the eigenvalue spectrum of a certain matrix associated with this graph. Note that our algorithm is not wholly complete but only complete with high probability; in return for sacrificing completeness, we get small-size proofs that are efficiently computable. Note that the eigenvalue spectrum can be approximated to arbitrary accuracy in polynomial time by standard linear algebra methods. With respect to odd k we focus on the case $k = 3$ and show by extending the previous arguments that random 3-SAT instances with $n^{(3/2)+\varepsilon}$ clauses can be efficiently certified as unsatisfiable with high probability, again improving the $n^2/\log n$ bound of Beame et al. In very recent work the aforementioned results have been improved; for $k = 4$ we have an efficient certification algorithm for Cn^2 random clauses, where C is a sufficiently large constant [CoGoLaSch 2004]. And for $k = 3$, we have a bound of $\text{POLY}(\log n) \cdot n^{3/2}$ clauses [CoGoLa 2004]. It seems to be a very hard task to get a bound below $n^{k/2}$ random k -clauses; for one thing, this $n^{k/2}$ seems to be a natural barrier to the spectral techniques we use. For $k = 3$, [FeOf 2004] recently showed that formulas with $n^{3/2}$ random clauses can be efficiently certified as unsatisfiable.

Eigenvalues are used in two ways in the algorithmic theory of random structures: first, they can be used to find a solution of an NP-hard problem in a random instance generated in such a way that it has a solution (not known to the algorithm). An example for 3-colorability is shown in [AlKa97]. Second, they can be used to prove the absence of a solution of an NP-problem. However, these applications are somewhat rare at the moment. The most prominent example is the expansion property of random regular graphs [AlSp92]. Note that the expansion property is coNP-complete [Bl et al81] and the eigenvalues certify the absence of a nonexpanding subset of vertices (which is the solution in this case). Our result is an example of the second kind and it might be worthwhile to investigate more systematically the existence of efficient algorithms for coNP-complete properties of random structures. A general overview on eigenvalues as applied to random graphs is given in [Al98].

We use the following notation throughout. The probabilistic model $\text{Form}_{n,k,m}$ of k -CNF formulas with m clauses over n propositional variables is defined as follows: the probability space of clauses of size k , $\text{Clause}_{n,k}$, is the set of ordered k -tuples of literals over n propositional variables v_1, \dots, v_n . We write $l_1 \vee \dots \vee l_k$ with $l_i = x$ or $l_i = \neg x$, where x is one of our variables. To simplify the subsequent presentation, our definition of $\text{Clause}_{n,k}$ allows for clauses containing the same literal twice and

clauses containing a variable and its negation. We consider $\text{Clause}_{n,k}$ as endowed with the uniform probability distribution: the probability of a clause is given by $\text{Pr}[l_1 \vee \dots \vee l_k] = (1/(2n))^k$. $\text{Form}_{n,k,m}$ is the m -fold cartesian product space of $\text{Clause}_{n,k}$. We write $F = C_1 \wedge \dots \wedge C_m$ and $\text{Pr}[F] = (1/(2n))^{k \cdot m}$. The probability space $\text{Form}_{n,k,p}$ is obtained by the following generation procedure: throw each clause with probability p into the formula to be generated, and the probability of a given formula with m clauses is $p^m \cdot (1-p)^{(2n)^k - m}$. These two spaces $\text{Form}_{n,k,m}$ and $\text{Form}_{n,k,p}$ are essentially equivalent when $m = p \cdot (2n)^k$. There are several other ways of defining k -SAT probability spaces (for example, clauses might be sets of literals instead of sequences; tautological clauses could be forbidden). Although we have not closely checked the details, we feel confident we are in line with general usage to assume that our results can be transferred to these other spaces, too. Note that the clause size k always is fixed.

1. Even clause size.

1.1. From random formulas to random graphs. The following simple observation underlies our algorithm.

LEMMA 1. *If a propositional formula F in k -CNF over n variables is satisfiable, there exists a subset S of at least $n/2$ variables such that F has no all-positive clause $x_1 \vee \dots \vee x_k$ with $x_i \in S$ for all i or F has no all-negative clause $\neg x_1 \vee \dots \vee \neg x_k$ with $x_i \in S$ for all i .*

Proof. Let $\mathcal{A} : \{v_1, \dots, v_n\} \rightarrow \{0, 1\}$ be a satisfying assignment for F . \mathcal{A} sets at least $n/2$ variables to 0 (= false) or to 1 (= true). In the first case the set of variables set to 0 satisfies the lemma; otherwise the set of variables set to 1 does. \square

Our algorithm proves unsatisfiability by efficiently showing the nonexistence of a set S , as in the preceding lemma. To do this we translate the nonexistence of S into a graph theoretical condition. To this end we assign two graphs to a formula. Let $F \in \text{Form}_{n,k,m}$, where k is even, be given. The graph $G = G_F$ depends only on the sequence of all-positive clauses of F :

- The set of vertices of G is $V = V_F = \{x_1 \vee \dots \vee x_{k/2} \mid x_i \text{ a variable}\}$. We have $|V| = n^{k/2}$ and V is independent of F .
- The set of edges of G , $E = E_F$ is given as follows. For $x_1 \vee \dots \vee x_{k/2} \neq y_1 \vee \dots \vee y_{k/2}$ we have $\{x_1 \vee \dots \vee x_{k/2}, y_1 \vee \dots \vee y_{k/2}\} \in E$ iff $x_1 \vee \dots \vee x_{k/2} \vee y_1 \vee \dots \vee y_{k/2}$ (or $y_1 \vee \dots \vee y_{k/2} \vee x_1 \vee \dots \vee x_{k/2}$) is a clause of F . Note that it is possible that $|E| < m$ as clauses might induce no edge, or two clauses might induce the same edge. We do not allow for loops or multiple edges.

The graph H_F is defined in a totally analogous way for the all-negative clauses of F .

Recall that an *independent set* of a graph G is a subset of vertices W of G such that we have no edge $\{v, w\}$ in G , where both $v, w \in W$. The next lemma follows directly from Lemma 1, as a set S of variables induces $|S|^{k/2}$ vertices in G_F consisting only of variables from S .

LEMMA 2. *If $F \in \text{Form}_{n,k,m}$ is satisfiable, then G_F or H_F has an independent set W of vertices with*

$$|W| \geq (n/2)^{k/2} = (1/2)^{k/2} \cdot |V|.$$

Note that as k remains constant when n gets large, this is a constant fraction of all vertices of G_F . We need to show that the distribution of G_F is just the distribution of a usual random graph. To this end let $G_{n,m}$ be the probability space of random

graphs with n labeled vertices and m different edges. Each graph is equally likely; that is, the probability of G is

$$\Pr[G] = 1 / \binom{\binom{n}{2}}{m}.$$

LEMMA 3. (1) *Conditional on the event in $\text{Form}_{n,k,m}$ that $|E_F| = r$, the graph G_F is a random member of the space $G_{\nu,r}$, where $\nu = n^{k/2}$ is the number of vertices of G_F .* (2) *Let $\varepsilon > 0$; with high probability the number of edges of G_F is between $m \cdot (1/2)^k \cdot (1 - \varepsilon)$ and $m \cdot (1/2)^k \cdot (1 + \varepsilon)$.*

Proof. (1) The proof is a well-known (but sometimes forgotten) trick: let G be a graph with vertices V_F as defined above and with edge set E with $|E| = r$. The set of formulas inducing this edge set can be constructed as follows: (a) Consider m slots for clauses and pick $r' \geq r$ of these slots; fill each of these r' slots with an edge from E_F , such that each edge occurs at least once; (b) make each edge in these r' slots into an all-positive clause ($2^{r'}$ possibilities); (c) fill each of the remaining slots with a clause containing at least one negative literal or with a clause having the same first and second halves.

(2) The claim follows from the following statements, which we prove further below:

- Let $\varepsilon > 0$ be fixed. The number of all-positive clauses of $F \in \text{Form}_{n,k,m}$ is between $(1 - \varepsilon) \cdot (1/2)^k \cdot m$ and $(1 + \varepsilon) \cdot (1/2)^k \cdot m$ with high probability.
- The number of all-positive clauses, such as $x_1 \vee \dots \vee x_{k/2} \vee x_1 \vee \dots \vee x_{k/2}$, that is, with the same first and second halves, is $o(m)$.
- The unordered pairs of positions of a formula F in which we have all-positive clauses, which induce only one edge, that are pairs of clauses $x_1 \vee \dots \vee x_k$, $y_1 \vee \dots \vee y_k$, where $\{x_1 \vee \dots \vee x_{k/2}, x_{k/2+1} \vee \dots \vee x_k\} = \{y_1 \vee \dots \vee y_{k/2}, y_{k/2+1} \vee \dots \vee y_k\}$, are also $o(m)$ with high probability.

This implies the claim of the lemma, with the actual ε slightly lower than the ε from the first statement above because we have only $o(m)$ clauses, which induce no additional edge.

First statement above: This statement follows with Chernoff bounds because the probability that a clause at a fixed position is all-positive is $(1/2)^k$ and clauses at different positions are independent.

Second statement above: The probability that the clause at position i has the same first and second halves is $(1/2n)^{k/2}$. The expected number of such clauses in a random F is therefore $m \cdot (1/2n)^{k/2} = o(m)$.

Third statement above: We fix 2 slots of clauses $i \neq j$ of F . The probability that the clauses in these slots have the same set of first and second halves is bounded above by $(2/n)^k$ and the expected number of such unordered pairs is at most $m^2 \cdot (2/n)^k = O(m/n)$, provided $m = O(n^{k-1})$, which we can assume. Let X be the random variable counting the number of unordered pairs of positions with clauses with the same first and second halves and let $\varepsilon > 0$. Markov's inequality gives us

$$\Pr[X > n^\varepsilon \cdot EX] \leq EX / (n^\varepsilon \cdot EX) = 1/n^\varepsilon.$$

Therefore we get that with high probability $X \leq n^\varepsilon \cdot (m/n) = o(m)$. \square

Spectral considerations. Eigenvalues of matrices associated with general graphs are somewhat less common (at least in computer science applications) than those of regular graphs. The monograph [Ch97] is a standard reference for the general case. The easier regular case is dealt with in [AlSp92]. The necessary linear algebra details cannot all be given here. They are very well presented in the textbook [St88].

Let $G = (V, E)$ be an undirected graph (loopless and without multiple edges) with $V = \{1, \dots, n\}$ being a standard set of n vertices. For $0 < p < 1$ we consider the matrix $A = A_{G,p}$ as in [KrVu2002] and [Ju82] and define it as follows: The $(n \times n)$ -matrix $A = A_{G,p} = (a_{i,j})_{1 \leq i,j \leq n}$ has $a_{i,j} = 1$ iff $\{i, j\} \notin E$ and $a_{i,j} = -(1-p)/p = 1 - 1/p$ iff $\{i, j\} \in E$. In particular $a_{i,i} = 1$. As A is real valued and symmetric, A has n real eigenvalues when counting them with their multiplicities and allowing for 0 as an eigenvalue (necessary when the matrix is not of full rank). We denote these eigenvalues by $\lambda_1(A) \geq \lambda_2(A) \geq \dots \geq \lambda_n(A)$.

Recall that the *independence number* of G , denoted by $\alpha(G)$, is the size (= number of vertices) of a largest independent set of G . In general it is NP-hard to determine the independence number. But we have an efficiently computable bound as follows.

LEMMA 4 (Lemma 4 of [KrVu2002]). *For any p with $p > 0$ we have $\lambda_1(A_{G,p}) \geq \alpha(G)$.*

Proof. Let $l = \alpha(G)$. Let χ be the characteristic column vector of a largest independent set of G (i.e., taking the value 1 on elements from the set and 0 otherwise). The matrix $A_{G,p}$ has an $(l \times l)$ -block which contains only 1's. This block of course is indexed with the vertices from our largest independent set. From the Courant–Fisher characterization of the eigenvalues of real valued symmetric matrices we get that

$$\lambda_1(A) \geq \frac{\chi^{tr} \cdot A \cdot \chi}{\chi^{tr} \cdot \chi} = l^2/l = l. \quad \square$$

In order to bound the size of the eigenvalues of $A_{G,p}$ when G is a random graph we rely on a suitably modified version of the following theorem.

THEOREM 5 (Theorem 2 of [FuKo81]). *For $1 \leq i, j \leq n$, and $i \leq j$, let $a_{i,j}$ be independent, real valued random variables (not necessarily identically distributed) satisfying the following conditions where the values K, ν, σ are constants independent of n :*

- $|a_{i,j}| \leq K$ for all $i \leq j$,
- the expectation $Ea_{i,i} = \nu$ for all i ,
- the expectation $Ea_{i,j} = 0$ for all $i < j$,
- the variance $Va_{i,j} = E[a_{i,j}^2] - (Ea_{i,j})^2 = \sigma^2$ for all $i < j$.

For $j \geq i$ let $a_{j,i} = a_{i,j}$ and let $A = (a_{i,j})_{1 \leq i,j \leq n}$ be the random $(n \times n)$ -matrix defined by the $a_{i,j}$. Let the eigenvalues of A be $\lambda_1(A) \geq \lambda_2(A) \geq \dots \geq \lambda_n(A)$. With probability at least $1 - (1/n)^{10}$ the matrix A is such that

$$\max\{|\lambda_i(A)| \mid 1 \leq i \leq n\} = 2 \cdot \sigma \cdot \sqrt{n} + O(n^{1/3} \cdot \log n) = 2 \cdot \sigma \cdot \sqrt{n} \cdot (1 + o(1)).$$

We intend to apply this theorem to the random matrix $A = A_{G,p}$, where G is a random graph from the probability space $G_{n,m}$. However, in this case the entries of A are not strictly independent and Theorem 5 cannot be directly applied. We first consider random graphs from the space $G_{n,p}$ and proceed to $G_{n,m}$ later on. Recall that a random graph G from $G_{n,p}$ is obtained by inserting each possible edge with probability p independently of other edges.

For p constant and G a random member from $G_{n,p}$ the assumptions of Theorem 5 can easily be checked to apply to $A_{G,p}$. However, for sparser random graphs, that is, $p = p(n) = o(1)$, the situation changes. We have that $a_{i,j}$ can assume the value $-1/o(1) + 1$ and thus its absolute value is no longer bounded above by a constant. The same applies to the variance $\sigma^2 = (1-p)/p = 1/o(1) - 1$.

However, it can be checked that the proof of Theorem 5 as given in [FuKo81] holds as long as we consider matrices $A_{G,p}$, where $p = (\ln n)^7/n$. In this case we

have that $K = n/(\ln n)^7 - 1$ and $\sigma^2 = n/(\ln n)^7 - 1$. With this modification and the previous assumptions, the proof of [FuKo81] leads to the following.

COROLLARY 6. *With probability at least $1 - (1/n)^{10}$ the random matrix A satisfies*

$$\max\{|\lambda_i(A)| \mid 1 \leq i \leq n\} = 2 \cdot \sigma \cdot \sqrt{n} + O(n/(\ln n)^{22/6}) = 2 \cdot (1/(\ln n)^{7/2}) \cdot n \cdot (1 + o(1)).$$

Proof. We sketch the changes, which need to be applied to the proof of Theorem 2 in [FuKo81]. These changes refer to the final estimates of the proof on page 237 of [FuKo81]. We set

$$k := (\sigma/K)^{1/3} \cdot n^{1/6} = (\ln n)^{7/6}(1 + o(1));$$

in fact, k should be the following even number: we set the error term

$$\nu := 50 \cdot n/(\ln n)^{22/6}.$$

We have

$$2 \cdot \sigma \cdot \sqrt{n} = 2 \cdot n/(\ln n)^{7/2} = 2 \cdot n/(\ln n)^{21/6},$$

which implies that $\nu = o(2 \cdot \sigma \cdot \sqrt{n})$. Concerning the error estimate, we get

$$\frac{\nu \cdot k}{2 \cdot \sigma \cdot \sqrt{n} + \nu} = \frac{50 \cdot (\ln n)^{7/6}}{(\ln n)^{1/6}} \cdot (1 + o(1)) = 50 \cdot \ln n \cdot (1 + o(1)).$$

This implies the claim. \square

Corollary 6 together with Lemma 4 gives us an efficiently computable certificate bounding the size of independent sets in random graphs from $G_{n,m}$.

COROLLARY 7. *Let G be a random member from $G_{n,m}$, where $m = ((\ln n)^7/2) \cdot n$, and let $p = m/\binom{n}{2} = (\ln n)^7/(n-1)$. We have with high probability that*

$$\lambda_1(A_{G,p}) \leq 2 \cdot (1/(\ln n)^{7/2}) \cdot n \cdot (1 + o(1)).$$

Proof. The proof is a standard transfer from the random graph model $G_{n,p}$ to $G_{n,m}$. For G random from $G_{n,p}$ the induced random matrix $A_{G,p}$ satisfies the assumptions of the last corollary. We have that with probability at least $1 - (1/n)^{10}$ the eigenvalues of $A_{G,p}$ are bounded by $2 \cdot (1/(\ln n)^{7/2}) \cdot n \cdot (1 + o(1))$.

By the de Moivre–Laplace local limit theorem for the binomial distribution the probability that a random graph from $G_{n,p}$ has *exactly* m edges is of $\Omega(1/(n \cdot p)^{1/2}) = \Omega(1/(\ln n)^{7/2})$. This implies the claim, as the probability in $G_{n,p}$ that the eigenvalue is not bounded as claimed is $O((1/n)^{10}) = o(1/(\ln n)^{7/2})$. (We omit the formal conditioning argument.) \square

1.2. The algorithm. We fix the clause size $k \geq 4$ and assume that k is even. We consider the probability space of formulas $\text{Form} = \text{Form}_{n,k,m}$, where the number of clauses is

$$m = 2^k \cdot (\ln n^{k/2})^7 \cdot n^{k/2} = 2^k \cdot (k/2)^7 \cdot (\ln n)^7 \cdot n^{k/2}.$$

Given a random formula F from Form the algorithm first considers the all-positive clauses from F and constructs the graph G_F . From Lemma 3 we know that $G = G_F$ is a random member of $G_{\nu,\mu}$, where $\nu = n^{k/2}$ and μ is at least $m \cdot (1/2)^k \cdot (1 - \varepsilon) =$

$(\ln \nu)^7 \cdot \nu \cdot (1 - \varepsilon)$, where we fix $\varepsilon > 0$ sufficiently small—in fact, $\varepsilon = 1/2$ will do. In the case when the number of edges is smaller than this bound, the algorithm fails.

The algorithm determines the matrix $A = A_{G,p}$, where $p = \mu/\binom{\nu}{2} \geq (\ln \nu)^7/(\nu - 1)$. From Corollary 7 we get that with high probability

$$\lambda_1(A) \leq 2 \cdot (1/(\ln \nu)^{7/2}) \cdot \nu \cdot (1 + o(1)),$$

and thus $\lambda_1(A) < (1/2)^k \cdot \nu = (1/2)^k \cdot n^{k/2}$ with high probability. The algorithm approximates $\lambda_1(A)$ by a polynomial time algorithm. In the case when the preceding event does not occur, the algorithm fails. By Lemma 4 G_F is now certain (not only certain with high probability) to have no independent set comprising $(1/2)^k \cdot n^{k/2}$ vertices.

The algorithm proceeds in the same way for the all-negative clauses and the graph H_F . When it succeeds (which happens with high probability) we have that F is unsatisfiable by Lemma 2. If the algorithm fails, which happens with probability $o(1)$, we do not know whether the formula is satisfiable or not.

When the number of literals k is odd we can extend each clause by a random literal and apply the preceding algorithm. It succeeds with high probability when the number of clauses is

$$2^{k+1} \cdot ((k + 1)/2)^7 \cdot (\ln n)^7 \cdot n^{(k+1)/2}.$$

Concerning the probability space $\text{Form}_{n,k,p}$, the algorithm succeeds with high probability when p is picked such that the expected number of clauses is at least $(1 + \varepsilon) \cdot$ “the bound above” because, in this case, the number of clauses is with high probability at least as large as this bound.

2. Clause size 3. From now on we restrict our attention to the family of probability spaces $\text{Form}_{n,p} = \text{Form}_{n,3,p}$. We assume that $p = p(n) = 1/n^{1+\gamma}$, where $1/2 > \gamma > 0$ is a constant. Our formulas get sparser with increasing γ . Note that our space of formulas is analogous to the space of random graphs $G_{n,p}$. The number of clauses in a random instance from $\text{Form}_{n,p}$ follows the binomial distribution with parameters $8n^3$ and p , $\text{Bin}(8n^3, p)$, and the expected number of clauses is $8n^3 \cdot p = 8 \cdot n^{2-\gamma} = 8 \cdot n^{3/2+\varepsilon}$, where $\varepsilon = 1/2 - \gamma > 0$.

2.1. From random 3-SAT instances to random graphs. We state a graph theoretical condition which implies the unsatisfiability of a 3-SAT instance F over n propositional variables. To this end we again define the graphs G_F and H_F . $G_F = (V_F, E_F)$ is defined as follows:

- V_F is the set of ordered pairs over the n propositional variables. We have $|V_F| = n^2$.
- The edge $(a_1, b_1) \text{---} (a_2, b_2)$ (where in order to avoid loops $(a_1, b_1) \neq (a_2, b_2)$, that is, $a_1 \neq a_2$ or $b_1 \neq b_2$) is in E_F iff there exists a variable z such that F contains the two clauses $a_1 \vee a_2 \vee z$ and $b_1 \vee b_2 \vee \neg z$.

The graph H_F is defined analogously but with different clauses: Its vertices are, as before, ordered pairs of variables, and $(a_1, b_1) \text{---} (a_2, b_2)$ is an edge iff F has the clauses $\neg a_1 \vee \neg a_2 \vee z$ and $\neg b_1 \vee \neg b_2 \vee \neg z$ for a variable z . Note that in the case of G_F the clause pairs $a_1 \vee a_2 \vee z, b_1 \vee b_2 \vee \neg z$ and $a_2 \vee a_1 \vee z', b_2 \vee b_1 \vee \neg z'$ induce the same edge $(a_1, b_1) \text{---} (a_2, b_2)$. Of course analogous remarks apply to H_F .

Some remarks concerning the intuition of this definition follow from the previous case, $k = 4$. The clause $a_1 \vee a_2 \vee b_1 \vee b_2$ is obtained by resolution [Sch89] with z from the two clauses $a_1 \vee a_2 \vee z$ and $b_1 \vee b_2 \vee \neg z$, which define an edge of G_F . Similarly we

have that $\neg a_1 \vee \neg a_2 \vee \neg b_1 \vee \neg b_2$ is obtained from $\neg a_1 \vee \neg a_2 \vee z$ and $\neg b_1 \vee \neg b_2 \vee \neg z$. The correctness of resolution states that F is unsatisfiable if a set of resolvents (that is, clauses obtained by resolution) of F is unsatisfiable.

For any z the number of clauses such as $a_1 \vee a_2 \vee z$ and $b_1 \vee b_2 \vee \neg z$ is concentrated at the expectation $\approx n^2 \cdot p = n^{1-\gamma} > n^{1/2}$ as $\gamma < 1/2$. Applying resolution with z to all these clauses gives $\approx n^{(1-\gamma)^2} > n$ clauses $a_1 \vee a_2 \vee b_1 \vee b_2$. Doing this for all n variables z gives $> n^2$ all-positive clauses of size 4. In the same way we get $> n^2$ all-negative 4-clauses. Efficiently bounding the size of independent sets in these graphs, we get an efficient algorithm which demonstrates unsatisfiability of these newly obtained 4-clauses. The correctness of resolution implies that F itself is unsatisfiable.

Some detailed remarks concerning G_F follow: Only for technical reasons is the variable z , which is resolved, the *last* variable in our clauses. (Recall we consider clauses as ordered triples.) More important is the fact that the edge reflects the resolvent $a_1 \vee a_2 \vee b_1 \vee b_2$ *not* in the most natural way by the edge $(a_1, a_2) \text{---} (b_1, b_2)$ but by $(a_1, b_1) \text{---} (a_2, b_2)$. The variables of the vertices connected by the edge come from the *different* clauses taking part in the resolution step. This is important in decreasing the stochastic dependency of the edges of G_F when F is a random formula. Again we have the convention (more of a technical nature) that the variables in the first position of each vertex come from the clause which contains the positive literal z , whereas the second variables b_1, b_2 come from the clause with $\neg z$.

Recall again that $\alpha(G)$ is the *independence number* of G , that is, the maximum number of vertices of an independent set of G .

THEOREM 8. *If F is a 3-SAT instance over n variables which is satisfiable, then we have*

$$\alpha(G_F) \geq n^2/4 \quad \text{or} \quad \alpha(H_F) \geq n^2/4.$$

Proof. Let \mathcal{A} be an assignment of the n underlying propositional variables with 0, 1 (where 0 = false and 1 = true), which makes F true. We assume that \mathcal{A} assigns 1 to at least $n/2$ variables. Let S be this set of variables. We show that the set of vertices $S \times S$ is an independent set of H_F . As this set has at least $(n/2)^2 = n^2/4$ vertices, the claim holds.

Let $(a_1, b_1) \text{---} (a_2, b_2)$ be an arbitrary edge from H_F . Then F has the clauses $\neg a_1 \vee \neg a_2 \vee z$ and $\neg b_1 \vee \neg b_2 \vee \neg z$ (or $\neg a_2 \vee \neg a_1 \vee z$ and $\neg b_2 \vee \neg b_1 \vee \neg z$). As the assignment \mathcal{A} makes F true, \mathcal{A} sets at least one of the literals $\neg a_1$, $\neg a_2$, $\neg b_1$, or $\neg b_2$ to 1. (Here the correctness proof of resolution is hidden: The clause $\neg a_1 \vee \neg a_2 \vee \neg b_1 \vee \neg b_2$ is a resolvent of $\neg a_1 \vee \neg a_2 \vee z$ and $\neg b_1 \vee \neg b_2 \vee \neg z$ and we have that if \mathcal{A} satisfies F , then \mathcal{A} satisfies all resolvents of F .) So \mathcal{A} sets at least one of the variables a_1 , a_2 , b_1 , or b_2 to 0. But this means that one of these variables is not in our set S . This finishes our proof, as now $(a_1, b_1) \notin S \times S$ or $(a_2, b_2) \notin S \times S$. As the initially chosen edge is arbitrary, we get that $S \times S$ is an independent set of H_F . Finally, if \mathcal{A} sets at least half of the variables to 0, we apply the same argument to G_F . \square

Now we will proceed in an analogous way as before: Given a random F from $\text{Form}_{n,p}$ the graphs G_F and H_F are certain random graphs. With high probability our algorithm certifies that G_F has no independent set of size $\geq n^2/4$. The same applies to H_F . Therefore F is certified unsatisfiable.

The occurrence of different edges in G_F and H_F is not fully independent and techniques from the area of standard random graphs cannot be applied without further consideration. From now on we restrict our attention to G_F ; of course everything applies also to H_F . We collect some basics about G_F .

An edge $(a_1, b_1) \text{---} (a_2, b_2)$ in G_F is only possible if $a_1 \neq a_2$ or $b_1 \neq b_2$. We take a look at the structure of the clause sets which induce the fixed edge $(a_1, b_1) \text{---} (a_2, b_2)$. The edge $(a_1, b_1) \text{---} (a_2, b_2)$ is in G_F iff F contains at least one of the pairs of clauses $a_1 \vee a_2 \vee z$ and $b_1 \vee b_2 \vee \neg z$ or one of the pairs $a_2 \vee a_1 \vee z$ and $b_2 \vee b_1 \vee \neg z$ for a variable z .

Case 1. $a_1 \neq a_2$ and $b_1 \neq b_2$. In this case all z -clauses are different and all $\neg z$ -clauses are different, too. As the z and $\neg z$ clauses are clearly different for each z , we have $2n$ disjoint pairs of clauses which induce the edge $(a_1, b_1) \text{---} (a_2, b_2)$.

Case 2. $a_1 = a_2$ and $b_1 \neq b_2$. In this case the clauses $a_1 \vee a_2 \vee z$ are all different. However, $a_1 \vee a_2 \vee z = a_2 \vee a_1 \vee z$. The $\neg z$ -clauses are all different, as are the z - and $\neg z$ -clauses. We have altogether $2n$ pairs of clauses, where two pairs always have the common clause $a_1 \vee a_2 \vee z$.

The last case, $a_1 \neq a_2$ and $b_1 = b_2$, is analogous to the second case.

With these observations we can get a first impression of the probability of a fixed edge in G_F : If $a_1 \neq a_2$ and $b_1 \neq b_2$, the number of pairs of clauses which induce the edge $(a_1, b_1) \text{---} (a_2, b_2)$ is distributed as $\text{Bin}(2n, p^2)$. The probability that the edge is induced by two pairs of clauses is at most $\binom{2n}{2} \cdot p^4 = o(2np^2)$. This makes it intuitively clear that the probability of $(a_1, b_1) \text{---} (a_2, b_2)$ being in G_F is about $2n \cdot p^2$.

If $a_1 = a_2$ and $b_1 \neq b_2$, we have that the number of clauses such as $b_1 \vee b_2 \vee \neg z$ or $b_2 \vee b_1 \vee \neg z$ is distributed as $\text{Bin}(2n, p)$. The probability of having at least two of these clauses is $O(n^2 p^2) = o(2np)$. Restricting ourselves to the occurrence of at least one of these clauses, it becomes intuitively clear that the probability of the edge $(a_1, b_1) \text{---} (a_2, b_2)$ should also be about $2n \cdot p^2$.

LEMMA 9. We fix the edge $e = (a_1, b_1) \text{---} (a_2, b_2)$.

(a) For $a_1 \neq a_2$ and $b_1 \neq b_2$ we have that

$$\Pr[F; e \text{ is an edge of } G_F] = 2n \cdot p^2 \cdot \left(1 + O\left(\frac{1}{n^{1+2\gamma}}\right)\right).$$

(b) For $a_1 = a_2$ and $b_1 \neq b_2$ this probability is

$$2n \cdot p^2 \cdot \left(1 + O\left(\frac{1}{n^{1+\gamma}}\right)\right).$$

The same applies of course to $a_1 \neq a_2$ and $b_1 = b_2$.

Proof. (a) Recalling the considerations just before this lemma, we have for the probability that G_F has the edge e

$$\Pr[F; e \text{ is an edge of } G_F] = 1 - (1 - p^2)^{2n}.$$

Using the binomial formula and further simple estimates estimates such as

$$\binom{2n}{i} (-p^2)^i \leq (2np^2)^i,$$

we get

$$1 - (1 - p^2)^{2n} = 1 - 1 + 2n \cdot p^2 - \sum_{i=2}^{2n} \binom{2n}{i} (-p^2)^i \geq 2n \cdot p^2 - \frac{4}{n^{2+4\gamma}} \cdot \sum_{i \geq 0} \left(\frac{2}{n^{1+2\gamma}}\right)^i.$$

In the same way,

$$1 - (1 - p^2)^{2n} \leq 2n \cdot p^2 + \frac{4}{n^{2+4\gamma}} \cdot \sum_{i \geq 0} \left(\frac{2}{n^{1+2\gamma}}\right)^i$$

and the convergence of the geometric series and $2n \cdot p^2 = 2/n^{1+2\gamma}$ imply the claim.

(b) We have that the edge e is not in G_F iff for no z it holds that F has the clause $a_1 \vee a_2 \vee z$ and one of the clauses $b_1 \vee b_2 \vee \neg z$ or $b_2 \vee b_1 \vee \neg z$. For a given z we get that $\Pr[F; F \text{ has } a_1 \vee a_2 \vee z \text{ and one of } b_1 \vee b_2 \vee \neg z, b_2 \vee b_1 \vee \neg z] = p \cdot (2p - p^2) = 2p^2 \cdot (1 - (p/2))$. As these triples of clauses are all disjoint for different z , we get that the probability that the edge e is not in G_F is $(1 - 2p^2 \cdot (1 - (p/2)))^n$ and

$$\begin{aligned} & \Pr[e \text{ is in } G_F] \\ &= 1 - \left(1 - 2p^2 \cdot \left(1 - \frac{p}{2}\right)\right)^n \geq n \cdot 2p^2 \cdot \left(1 - \frac{p}{2}\right) - \frac{4}{n^{2+4\gamma}} \sum_{i \geq 0} \left(\frac{2}{n^{1+2\gamma}}\right)^i \\ &= 2n \cdot p^2 \cdot \left(1 + O\left(\frac{1}{n^{1+\gamma}}\right)\right). \end{aligned}$$

In the same way we get the upper bound

$$\Pr[F; e \text{ is in } G_F] \leq 2n \cdot p^2 \cdot \left(1 + O\left(\frac{1}{n^{1+\gamma}}\right)\right)$$

and the claim holds. \square

The preceding lemma implies the following expectations.

COROLLARY 10. (a) Let the random variable X denote the number of edges of G_F . For the expectation of X we get

$$EX = n^{3-2\gamma} \cdot \left(1 + O\left(\frac{1}{n}\right)\right).$$

(b) Let $X_{(a_1, b_1)}$ be the degree of the vertex (a_1, b_1) in G_F . Then

$$E[X_{(a_1, b_1)}] = 2n^{1-2\gamma} \cdot \left(1 + O\left(\frac{1}{n}\right)\right).$$

Proof. (a) The expected number of edges such as $(a_1, b_1) - (a_2, b_2)$ with $a_1 \neq a_2$ and $b_1 \neq b_2$ in G_F is with Lemma 9(a) as $2n \cdot p^2 = 2/n^{1+2\gamma}$,

$$\begin{aligned} & \frac{n^2 \cdot (n-1)^2}{2} \cdot \frac{2}{n^{1+2\gamma}} \cdot \left(1 + O\left(\frac{1}{n^{1+2\gamma}}\right)\right) \\ &= (n^{3-2\gamma} - 2n^{2-2\gamma} + n^{1-2\gamma}) \cdot \left(1 + O\left(\frac{1}{n^{1+2\gamma}}\right)\right) = n^{3-2\gamma} \cdot \left(1 + O\left(\frac{1}{n}\right)\right) \end{aligned}$$

as $\gamma > 0$. For the expected number of edges $(a_1, b_1) - (a_2, b_2)$, where $a_1 = a_2$ (and $b_1 \neq b_2$), we get by Lemma 9(b)

$$\begin{aligned} & \frac{n^2 \cdot (n-1)}{2} \cdot \frac{2}{n^{1+2\gamma}} \cdot \left(1 + O\left(\frac{1}{n^{1+\gamma}}\right)\right) = (n^{2-2\gamma} - n^{1-2\gamma}) \cdot \left(1 + O\left(\frac{1}{n^{1+\gamma}}\right)\right) \\ &= n^{2-2\gamma} \cdot \left(1 + O\left(\frac{1}{n}\right)\right). \end{aligned}$$

The same applies of course to these edges when $a_1 \neq a_2$ and $b_1 = b_2$. As $n^{2-2\gamma} = n^{3-2\gamma} \cdot 1/n$, the claim follows.

(b) Fixing the vertex (a_1, b_1) , we see the number of edges $(a_1, b_1) \text{---} (a_2, b_2)$ with $a_1 \neq a_2$ and $b_1 \neq b_2$ altogether is $(n - 1)^2$, and for the expected number of such edges we get, as $2n \cdot p^2 = 2/n^{1+2\gamma}$,

$$\begin{aligned} & (n - 1)^2 \cdot \frac{2}{n^{1+2\gamma}} \cdot \left(1 + O\left(\frac{1}{n^{1+2\gamma}}\right)\right) \\ = & \left(2n^{1-2\gamma} - \frac{4}{n^{2\gamma}} + \frac{2}{n^{1+2\gamma}}\right) \cdot \left(1 + O\left(\frac{1}{n^{1+2\gamma}}\right)\right) = 2n^{1-2\gamma} \cdot \left(1 + O\left(\frac{1}{n}\right)\right). \end{aligned}$$

The number of edges $(a_1, b_1) \text{---} (a_2, b_2)$, where $a_1 = a_2$ and $b_1 \neq b_2$ altogether, is $n - 1$, and for the expected number of these edges we get with Lemma 9(b)

$$(n - 1) \cdot \frac{2}{n^{1+2\gamma}} \cdot \left(1 + O\left(\frac{1}{n^{1+\gamma}}\right)\right) = O\left(\frac{1}{n^{2\gamma}}\right) = n^{1-2\gamma} \cdot O\left(\frac{1}{n}\right).$$

Of course the same applies to these edges, where $a_1 \neq a_2$ and $b_1 = b_2$. \square

Observe that $n^2 \cdot 2n^{1-2\gamma} = 2 \cdot n^{3-2\gamma}$, reflecting the fact that the sum of the degrees of all vertices is two times the number of edges. The number of vertices is n^2 and the probability that a given edge occurs is $\approx 2/n^{1+2\gamma}$. Disregarding edge dependencies, we see G_F is a random graph $G_{n^2, p'}$, where $p' \approx 2/n^{1+2\gamma} = 2n^{1-2\gamma}/n^2$. Note that the exponent is $1 - 2\gamma > 0$ as $0 < \gamma < 1/2$. The analogous situation in standard random graphs is the probability space $G_{n, p'}$, where $p' = n^\epsilon/n$.

For random graphs according to $G_{n, p'}$ the degree of any vertex is with high probability concentrated at its expectation. This follows easily from tail bounds for the binomial distribution. In the case of our graphs G_F and H_F there is possibly some overlap between clauses inducing edges. This entails a weak kind of stochastic dependency between the occurrence of different edges. Nevertheless the following concentration result holds.

THEOREM 11. *With high probability we have for all vertices (a_1, b_1) of G_F that the degree of (a_1, b_1) is of $2 \cdot n^{1-2\gamma}(1 + o(1))$.*

Proof. Let the variables a_1, b_1 be given. For a variable z let X_z be the indicator random variable on the probability space $\text{Form}_{n, p}$ of the event that there are (non-negated) variables a_2, b_2 such that $a_1 \vee a_2 \vee z \in F$ and $b_1 \vee b_2 \vee \neg z \in F$. Let Y_z indicate the event, $a_2 \vee a_1 \vee z \in F$, and $b_2 \vee b_1 \vee \neg z \in F$. Let $X = \sum X_z$ and Y be the same for the Y_z . We show two points: The degree of the vertex (a_1, b_1) of G_F is with probability at least $1 - o(1/n^2)$ equal to $(X + Y) \cdot (1 + o(1))$. Moreover, with probability $1 - o(1/n^2)$ the random variable $X + Y$ is $2 \cdot n^{1-2\gamma}(1 + o(1))$.

First we compute the value of $X + Y$. For a given variable z we have

$$\Pr[X_z = 1] = (1 - (1 - p)^n) \cdot (1 - (1 - p)^n) = 1 - 2 \cdot (1 - p)^n + (1 - p)^{2n},$$

as $1 - (1 - p)^n$ is the probability that at least for one a_2 the clause $a_1 \vee a_2 \vee z$ occurs in a random formula. As $0 \leq \gamma < 1/2$, we get from the binomial theorem and the formula for the geometric series that

$$(1 - p)^n = 1 - 1/n^\gamma + 1/(2 \cdot n^{2\gamma}) + O(1/n^{3\gamma})$$

and

$$(1 - p)^{2n} = 1 - 2/n^\gamma + 2/n^{2\gamma} + O(1/n^{3\gamma}).$$

Plugging these values into the formula for $Pr[X_z = 1]$, we get

$$Pr[X_z = 1] = 1/n^{2\gamma} + O(1/n^{3\gamma}) = 1/n^{2\gamma} \cdot (1 + o(1)),$$

as we can assume that $\gamma > 0$.

As distinct X_z refer to disjoint sets of clauses, the X_z are stochastically independent and X follows the binomial distribution $\text{Bin}(n, 1/n^{2\gamma} \cdot (1 + o(1)))$. Therefore the expectation of X is $n^{1-2\gamma} \cdot (1 + o(1))$. As $\gamma < 1/2$, we have $1 - 2\gamma > 0$, and exponential tail bounds for the binomial distribution imply that the random variable X is with probability at least $1 - o(1/n^2)$ of $n^{1-2\gamma} \cdot (1 + o(1))$. Of course everything holds in the same way for Y , and the required concentration result for $X + Y$ holds.

We come to the degree. The degree of (a_1, b_1) in the graph G_F is the number of distinct vertices $(a_2, b_2) \neq (a_1, b_1)$ for which there is a variable z such that $a_1 \vee a_2 \vee z \in F$ and $b_1 \vee b_2 \vee \neg z \in F$ or $a_2 \vee a_1 \vee z \in F$ and $b_2 \vee b_1 \vee \neg z \in F$. We show that there exists a constant $C = C(\gamma)$ such that with probability at most $o(1/n^2)$ the degree of (a_1, b_1) differs additively from the random variable $X + Y$ by more than this constant C . We need to analyze the cases in which a variable z with $X_z = 1$ or $Y_z = 1$ induces either no or strictly more than one additional edge incident with (a_1, b_1) .

Case 1. We have $X_z = 1$ but no additional edge is induced. This can happen only if $X_z = 1$ due to the clauses $a_1 \vee a_1 \vee z, b_1 \vee b_1 \vee \neg z \in F$ or if the edge induced by $a_1 \vee a_2 \vee z, b_1 \vee b_2 \vee \neg z \in F$ is also induced by another set of two clauses from F .

The first case occurs for at most C variables z with probability $1 - o(1/n^2)$. For constant C the expected number of sets of C variables z such that the two clauses above are in a random F is at most

$$n^C \cdot (1/n^{1+\gamma})^{2C} = 1/n^{(1+2\gamma) \cdot C}$$

and for $C = 2$, recalling $\gamma > 0$, this expectation is $o(1/n^2)$. By Markov's inequality the probability that there are at least 2 variables z accompanied by clauses as above in a random formula F is $o(1/n^2)$. The case $Y_z = 1$ proceeds in the same way.

The second case is slightly more complex but follows with the same principle. First we consider the case that no additional edge is induced by the clauses $a_1 \vee a_2 \vee z, b_1 \vee b_2 \vee \neg z \in F$ due to a *disjoint* set of two clauses, which yields the same edge. That is, there exists a $z' \neq z$ such that $a_1 \vee a_2 \vee z', b_1 \vee b_2 \vee \neg z' \in F$ or there is a z' ($z' = z$ may be possible) such that $a_2 \vee a_1 \vee z', b_2 \vee b_1 \vee \neg z' \in F$. For a suitably chosen constant C this situation occurs for C variables z with probability $o(1/n^2)$. The expected number of sets of C variables z such that for each of these z one of the preceding possibilities occurs in a random F is bounded above by

$$n^C \cdot n^C \cdot n^{2C} \cdot 2^C \cdot (1/n^{1+\gamma})^{4C} = 2^C / n^{4C\gamma}.$$

For $C \geq 1$ and $\gamma > 0$ this expectation is $o(1)$. Picking $C > 1/(2\gamma)$ makes this expectation $o(1/n^2)$ and the result follows with Markov's inequality.

It may also be the case that no additional edge is induced due to a set of two clauses which, however, now is *not disjoint* to the two clauses $a_1 \vee a_2 \vee z, b_1 \vee b_2 \vee \neg z \in F$. In this case we must have $a_1 = a_2$ (or the same for the b_i 's) and we have the clause $b_2 \vee b_1 \vee \neg z \in F$. Therefore we now have $Y_z = 1$. For the expectation as before we get

$$n^C \cdot n^{2C} \cdot 2^C \cdot (1/n^{1+\gamma})^{3C} = 1/n^{3C\gamma}$$

which is $o(1/n^2)$ for $C \geq 2/(3\gamma)$ and $\gamma > 0$.

Case 2. We consider the situation $X_z = 1$. Strictly more than one edge is induced by the clauses with $z \neg z$ in the third position. We show that the number of edges incident with (a_1, b_1) induced by clauses with the fixed $z, \neg z$ in the end can be bounded above by a constant. For the appropriate expectation we get

$$n^C \cdot n^{C'} \cdot 2^C \cdot 2^{C'} \cdot (1/n^{1+\gamma})^{C+C'} = O(1/n^{\gamma \cdot (C+C')})$$

and picking $C + C'$ large enough we get that at least $C \cdot C'$ edges are induced due the clauses with $z, \neg z$ in the end with probability $o(1/n^2)$.

We still need to bound the number of z 's altogether such that strictly more than one edge is induced due to clauses with $z, \neg z$ in the end. The appropriate expectation is

$$n^C \cdot n^{3C} \cdot 2^C \cdot (1/n^{1+\gamma})^{3C} = 1/n^{(3\gamma-1) \cdot C},$$

which is $o(1)$ if $\gamma > 1/3$, which we can safely assume, and $C \geq 1$. Picking $C > 2/(3\gamma - 1)$ we see the expectation is $o(1/n^2)$.

The claim of the theorem now follows from the preceding considerations. Picking C as the sum of all possible deviations from exactly one edge being induced by $X_z = 1$ or $Y_z = 1$ should do. \square

2.2. Spectral considerations. In this section we prove a general relationship between the size of an independent set in a graph and the eigenvalues of its adjacency matrix. Then we prove that the random graphs G_F and H_F satisfy certain eigenvalue bounds with high probability. These eigenvalue bounds certify that the graphs G_F and H_F do not have independent sets, which are necessary for F to be satisfiable by Theorem 8. Again background from spectral graph theory can be found for regular graphs in [AlSp92] and for the general case in [Ch97]. The linear algebra required is well presented in [St88].

Let $G = (V, E)$ be an undirected graph and $A = A_G$ the adjacency matrix of G . Let A 's eigenvalues be ordered as $\lambda_1(A) \geq \dots \geq \lambda_n(A)$, with $n = |V|$. We abbreviate $\lambda_i = \lambda_i(A)$. We say that G is ν -separated if $|\lambda_i| \leq \nu \lambda_1$ for $i > 1$. With $\lambda = \max_{i>1} |\lambda_i|$ this reads $\lambda \leq \nu \lambda_1$. We say that G is ϵ -balanced for some $\epsilon > 0$ if there is a real d such that the degree of each vertex is between $d(1 - \epsilon)$ and $d(1 + \epsilon)$. As opposed to Lemma 4, the subsequent theorem only uses the eigenvalues of the adjacency matrix of the graph considered.

THEOREM 12. *If G is ν -separated and ϵ -balanced, then G contains no independent set of size $> (n/5) + n \cdot f(\nu, \epsilon)$, where $f(\nu, \epsilon)$ tends to 0 as ν, ϵ tend to 0.*

We remark that this theorem can probably be greatly improved upon (see the remark in the proof). But this weak theorem does preclude independent sets of size $n/4$ for small ν, ϵ , and that is all we need here.

Proof. Let S be an independent subset of vertices of G . We will bound $|S|$. Let $T = V \setminus S$. Let χ_S, χ_T be the characteristic functions (represented as column vectors) of S, T , respectively, (i.e., taking the value 1 on the set and 0 outside of the set). As S is an independent set and G is ϵ -balanced, we have

$$(2.1) \quad d(1 - \epsilon)|S| \leq \left| \text{edges leaving } S \right| = \langle A_G \chi_S, \chi_T \rangle.$$

Note that $A_G \chi_S$ is the column vector whose i th entry is the number of edges going from vertex i into the set S . Recall that $T = V \setminus S$ and $\langle \dots, \dots \rangle$ is the standard inner product of two vectors. We show further below that

$$(2.2) \quad \langle A_G \chi_S, \chi_T \rangle \leq d(1 + \epsilon) \cdot (1/2 + \nu) \cdot \sqrt{|S||T|}.$$

Abbreviating $\theta = |S|/n$ we get from (2.1) and (2.2)

$$|S| \leq \frac{1+\varepsilon}{1-\varepsilon} \left(\frac{1}{2} + \nu \right) \cdot \sqrt{|S||T|}.$$

This implies that

$$\sqrt{\frac{|S|}{|T|}} \leq \frac{1+\varepsilon-\varepsilon+\varepsilon}{1-\varepsilon} \left(\frac{1}{2} + \nu \right) = \left(1 + \frac{2\varepsilon}{1-\varepsilon} \right) \left(\frac{1}{2} + \nu \right) = \frac{1}{2} + g(\varepsilon, \nu),$$

where $g(\varepsilon, \nu)$ goes to 0 when ε and ν do. Next we get

$$\frac{\theta}{1-\theta} \leq \left(\frac{1}{2} + g(\nu, \varepsilon) \right)^2 = \frac{1}{4} + g(\nu, \varepsilon) + g(\nu, \varepsilon)^2.$$

We set $f(\nu, \varepsilon) = (4/5)(g + g^2)$ and can easily conclude that

$$\begin{aligned} \theta &\leq (1-\theta) \cdot (1/4) + (1-\theta) \cdot g + (1-\theta) \cdot g^2 \leq (1/4) - \theta \cdot (1/4) + g + g^2 \\ &\Rightarrow \theta \leq (1/4)(4/5) + (4/5)(g + g^2) = 1/5 + f, \end{aligned}$$

which is the theorem.

We need to show inequality (2.2). Let u_1, \dots, u_n be an orthonormal basis of the n -dimensional vectorspace over the reals, where u_i is an eigenvector with eigenvalue λ_i of A_G . We can decompose the adjacency matrix as

$$A_G = \lambda_1 \cdot u_1 \cdot u_1^T + \lambda_2 \cdot u_2 \cdot u_2^T + \dots + \lambda_n \cdot u_n \cdot u_n^T,$$

where $u_i^T = (u_{i,1}, \dots, u_{i,n})$ is the transpose of the column vector u_i . Note that $\lambda_i \cdot (u_i \cdot u_i^T) \cdot v = \lambda_i \cdot v$ if $v = \alpha \cdot u_i$ and $\lambda_i \cdot (u_i \cdot u_i^T) \cdot v = 0$ for v orthogonal to u_i . Let

$$\mathcal{E} = A_G - \lambda_1 \cdot u_1 \cdot u_1^T = \sum_{i \geq 2} \lambda_i \cdot u_i \cdot u_i^T$$

and represent χ_S, χ_T over the basis of the u_i :

$$\chi_S = \sum_{i=1}^n \alpha_i \cdot u_i \quad \text{and} \quad \chi_T = \sum_{i=1}^n \beta_i \cdot u_i.$$

Recall the fact known as Parseval's equation:

$$|S| = \|\chi_S\|^2 = \sum \alpha_i^2 \quad \text{and} \quad |T| = \|\chi_T\|^2 = \sum \beta_i^2.$$

We get

$$\begin{aligned} \langle A_G \chi_S, \chi_T \rangle &= \langle \lambda_1 \cdot u_1 \cdot u_1^T \cdot \chi_S + \dots + \lambda_n \cdot u_n \cdot u_n^T \cdot \chi_S, \chi_T \rangle \\ &= \langle (\lambda_1 \cdot (u_1^T \cdot \chi_S)) \cdot u_1, \chi_T \rangle + \langle \mathcal{E} \cdot \chi_S, \chi_T \rangle \\ &= (\lambda_1 (u_1^T \cdot \chi_S)) \cdot (u_1^T \cdot \chi_T) + \langle \mathcal{E} \cdot \chi_S, \chi_T \rangle. \end{aligned}$$

We bound the two summands separately. Because of the orthornormality of the u_i we get

$$\begin{aligned} \langle \mathcal{E}\chi_S, \chi_T \rangle &= \langle \mathcal{E} \cdot (\alpha_1 u_1 + \dots + \alpha_n u_n), \chi_T \rangle \\ &= \langle \lambda_2 \alpha_2 u_2 + \dots + \lambda_n \alpha_n u_n, \beta_1 u_1 + \dots + \beta_n u_n \rangle \\ &= \langle \lambda_2 \alpha_2 u_2, \beta_2 u_2 \rangle + \langle \lambda_3 \alpha_3 u_3, \beta_3 u_3 \rangle + \dots + \langle \lambda_n \alpha_n u_n, \beta_n u_n \rangle \\ &= \lambda_2 \alpha_2 \beta_2 + \lambda_3 \alpha_3 \beta_3 + \dots + \lambda_n \alpha_n \beta_n \\ &\leq \lambda \cdot \sum_{i>1} |\alpha_i \beta_i| \leq \lambda \cdot \sqrt{\sum_{i>1} \alpha_i^2} \cdot \sqrt{\sum_{i>1} \beta_i^2} \leq \lambda \cdot \sqrt{\sum_{i \geq 1} \alpha_i^2} \cdot \sqrt{\sum_{i \geq 1} \beta_i^2} \\ &= \lambda \cdot \sqrt{|S|} \cdot \sqrt{|T|} \leq \nu \cdot d(1 + \varepsilon) \sqrt{|S||T|}, \end{aligned}$$

where the last step holds because λ_1 is bounded above by the maximal degree of a vertex, the step before the last is Parseval's equation, and the third step before the last is the Cauchy–Schwarz inequality, $\sum |\alpha_i \beta_i| \leq \sqrt{\sum \alpha_i^2} \cdot \sqrt{\sum \beta_i^2}$.

Now we come to the other summand, $(\lambda_1 (u_1^T \cdot \chi_S)) \cdot (u_1^T \cdot \chi_T)$. Let α, β be the average values of u_1 on S, T , respectively, that is, $\alpha = (\sum u_{1,j})/|S|$, where the sum goes over $j \in S$. With the Cauchy–Schwarz inequality we get

$$\alpha^2 = \frac{(\sum (u_{1,j} \cdot 1))^2}{|S|^2} \leq \frac{(\sum u_{1,j}^2) \cdot (\sum 1)}{|S|^2} = \frac{\sum u_{1,j}^2}{|S|}$$

which implies $\alpha^2 |S| \leq \sum u_{1,j}^2$. As $T = V \setminus S$ we get

$$\alpha^2 |S| + \beta^2 |T| \leq \sum_{j=1}^n u_{1,j}^2 = \|u_1\|^2 = 1.$$

Using the fact that the geometric mean is bounded by the arithmetic mean, this implies

$$\alpha \sqrt{|S|} \cdot \beta \sqrt{|T|} = \sqrt{\alpha^2 |S| \cdot \beta^2 |T|} \leq \frac{\alpha^2 |S| + \beta^2 |T|}{2} \leq \frac{1}{2}.$$

(The weakness of this theorem undoubtedly comes from the pessimistic first estimate, which is close to the truth only when $\alpha^2 |S|$ is close to $\beta^2 |T|$.) This implies, as λ_1 is bounded above by the maximum degree, that

$$\lambda_1 \cdot (u_1^T \chi_S) \cdot (u_1^T \chi_T) \leq d(1 + \varepsilon) \alpha |S| \cdot \beta |T| \leq (1/2) \cdot d(1 + \varepsilon) \sqrt{|S||T|},$$

and we get (2.2), finishing the proof. \square

We next show that the graphs G_F and H_F are ν -separated for a small ν . We do this by applying the trace method; see, for example, [Fr91]. For our purposes it is sufficient to use an elementary version of this method. We first give a general outline of this method and then apply it to the $G_{n,p}$ model of random graphs. Finally, we proceed to the technically more complex graphs G_F, H_F . For $A = A_G$, an adjacency matrix, we have from linear algebra that, for each $k \geq 0$, $\text{Trace}(A^k) = \sum_{i=1}^n \lambda_i^k$. (The trace of a matrix is the sum of the elements on the diagonal.) $\text{Trace}(A^k)$ can be calculated from the underlying graph as $\text{Trace}(A^k) = |\text{closed walks of length } k \text{ in the underlying graph}|$. A closed walk of length k in G is a walk like

$$a_0 \xrightarrow{e_1} a_1 \xrightarrow{e_2} a_2 \xrightarrow{e_3} \dots \xrightarrow{e_{k-1}} a_{k-1} \xrightarrow{e_k} a_k = a_0.$$

Note that the e_i and a_i need by no means be different, only $a_{i-1} \neq a_i$ as we assume the graph loopless. If k is even we have that all $\lambda_i^k \geq 0$ and we get $\text{Trace}(A^k) = \sum_{i=1}^n \lambda_i^k \geq \lambda_1^k + \max_{i>1} \lambda_i^k$. Abbreviating $\lambda = \max_{i>1} |\lambda_i|$ we get further

$$\lambda^k \leq \text{Trace}(A^k) - \lambda_1^k = \sum_{i=2}^n \lambda_i^k.$$

If the underlying adjacency matrix A is random, this applies in particular to the expected values:

$$(2.3) \quad E[\lambda^k] \leq E[\text{Trace}(A^k)] - E[\lambda_1^k] = E \left[\sum_{i=2}^n \lambda_i^k \right].$$

Now assume that k is an even constant, that n is a variable and sufficiently large (having a model that allows a variable n such as $G_{n,p}$), and that $E[\lambda^k] = o(\lambda_1^k)$ with high probability. Then we have that the graph underlying A is ν -separated for any fixed $\nu > 0$ with high probability, which is easy to see:

$$\Pr[\lambda > \nu \lambda_1] = \Pr[\lambda^k > (\nu \lambda_1)^k] = \Pr \left[\lambda^k > \frac{(\nu \lambda_1)^k}{E[\lambda^k]} \cdot E[\lambda^k] \right] \leq o(1),$$

where we apply Markov’s inequality, using the fact that $(\nu \lambda_1)^k / E[\lambda^k] = 1/o(1)$ with high probability, as k and ν are constant. The last estimate of course implies that for each $\nu > 0$ almost all graphs considered are ν -separated. (The idea of considering the k th power of the eigenvalues seems to be to increase the gap between the largest eigenvalue and the remaining eigenvalues.)

Now we apply this to the $G_{n,p}$ model of random graphs, where $p = n^\delta/n$ and $\delta > 0$ is a constant. We calculate the two expected values in (2.3) separately. The largest eigenvalue is always between the minimum and maximum degree of any vertex, as follows from the Courant–Fisher characterization of the eigenvalues of real symmetric matrices. From Chernoff bounds we know that for any $\varepsilon > 0$ with high probability all vertices x from a random G satisfy $(1 - \varepsilon)n^\delta \leq \text{degree of } x \leq (1 + \varepsilon)n^\delta$. Thus with high probability $(1 - \varepsilon)n^\delta \leq \lambda_1 \leq (1 + \varepsilon)n^\delta$. As the probability bounds of the degree estimate follow directly from Chernoff bounds, the probability that the estimate does not hold is exponentially low, that is, at most $n \cdot \exp(-\Omega(n^\delta))$. The same probability estimate applies to the estimate for λ_1 . (The exponentially small term is for a single fixed vertex; the factor n is necessary, as we have n vertices.) As k is even, $\lambda_1^k \geq k$ and

$$E[\lambda_1^k] \geq ((1 - \varepsilon)n^\delta)^k (1 - n \exp(-\Omega(n^\delta))) = (1 - \varepsilon)^k n^{\delta k} - o(1).$$

Next we come to the expectation of the trace in (2.3). For

$$\vec{a} = (a_0, \dots, a_{k-1}, a_k = a_0)$$

let $\text{walk}(\vec{a})$ be the indicator random variable of the event that the walk given by \vec{a} is possible in a random graph; that is, all edges $e_i = (a_{i-1}, a_i) = \{a_{i-1}, a_i\}$ for $1 \leq i \leq k$ occur. Then

$$E[\text{Trace}(A^k)] = E \left[\sum_{\vec{a}} \text{walk}(\vec{a}) \right] = \sum_{\vec{a}} P[\text{walk}(\vec{a}) = 1]$$

by linearity of expectation and as $\text{walk}(\vec{a})$ is an indicator random variable. To calculate the preceding sum we distinguish three types of possible walks \vec{a} . A walk is *distinct* iff all edges e_i are distinct. A walk is *duplicated* iff each edge among the e_i occurs at least twice. A walk is *quasi distinct* iff some edges among the e_i occur at least twice and some only once. (Notice that for quasi-distinct walks and duplicated walks, one type does not subsume the other—indeed, quasi-distinct walks must have an edge that occurs only once; furthermore the arguments given below differ essentially for the quasi-distinct and duplicated cases.) For \vec{a} distinct we have that $\text{Pr}[\text{walk}(\vec{a}) = 1] = p^k$, and the number of \vec{a} 's altogether which can induce a distinct walk is at most n^k because we can choose a_0, \dots, a_{k-1} . Hence, the expected number of distinct walks is bounded above by $n^{\delta k}$. (Compare our estimate for $E[\lambda_1]$.)

For \vec{a} duplicated we parametrize further with respect to the number j , with $1 \leq j \leq k/2$, of different edges among the e_i . Any walk which is possible at all is generated at least once by the following process: 1. Pick the j positions among k possible positions, where each of the j different edges occurs for the first time in $\vec{e} = (e_1, \dots, e_k)$: $\binom{k}{j} \leq k^k$ possibilities. 2. For each of the remaining $k - j$ positions specify which of the preceding first occurrences of an edge is to be used in this position: $\leq j^{k-j} \leq k^k$ possibilities. 3. Specify the vertices incident with the edges picked in 1. As the j edges from 1 must induce a connected graph, the number of possibilities here is at most n^{j+1} . Now we get for the expected number of duplicated walks

$$\begin{aligned} & \sum_{\vec{a} \text{ duplicated}} \text{Pr}[\text{walk}(\vec{a}) = 1] \\ & \leq \sum_{j=1}^{k/2} k^{2k} \cdot n^{j+1} \cdot (n^\delta/n)^j \leq \sum_{j=1}^{k/2} k^{2k} \cdot n \cdot n^{\delta j} \leq (k/2) \cdot k^{2k} \cdot n \cdot n^{\delta k/2}. \end{aligned}$$

Note that $1 + \delta k/2 < \delta k$ iff $2/\delta < k$. So for this estimate to be true k must increase if δ gets smaller.

For the number of quasi-distinct walks, we first assume that the last edge, e_k , is a unique edge of the walk. As our walks are quasi distinct, there are at most $1 \leq j \leq k - 2$ first occurrences of different edges in (e_1, \dots, e_{k-1}) . This implies that the expected number of quasi-distinct walks with the last edge unique is bounded by

$$\sum_{j=1}^{k-2} k^{2k} \cdot n^{j+1} \cdot \left(\frac{n^\delta}{n}\right)^j \cdot \frac{n^\delta}{n} \leq k \cdot k^{2k} \cdot n^{\delta(k-1)}.$$

We account for those quasi-distinct walks, where the last edge is not unique by shifting a unique edge to the end and counting as before. As there are k positions where a unique edge might occur, we need an additional factor of k . Note that always $\delta(k - 1) < \delta k$.

Summing the preceding estimates, we get

$$E[\text{Trace}(A^k)] \leq n^{\delta k} + k^2 \cdot k^{2k} \cdot (nn^{\delta k/2} + n^{\delta(k-1)}) = n^{\delta k} + o(n^{\delta k})$$

if $k > 2/\delta$ is an even constant, which we assume. Now (2.3) implies

$$E[\lambda^k] \leq (1 - (1 - \varepsilon)^k) \cdot n^{\delta k} + o(n^{\delta k})$$

as k even. As $\varepsilon > 0$ can be chosen arbitrarily small, k is constant and the preceding estimate holds whenever n is sufficiently large, we have that $E[\lambda^k] = o(n^{\delta k})$. Now

fix $\nu > 0$. By the general principle stated above we get that $Pr[\lambda > \nu\lambda_1] = o(1)$, meaning that almost all graphs are ν -separated. Applying Theorem 12 we can for almost all graphs from $G_{n,p}$ efficiently certify that they do not have independent sets with much more than $n/5$ vertices.

The treatment of our graphs G_F, H_F based on the method above is more technical but follows the same principles.

THEOREM 13. *For $F \in Form_{n,p,3}$ let $A = A_F$ be the adjacency matrix of G_F and let $\lambda_1 \geq \lambda_2 \geq \dots \geq \lambda_{n^2}$ be the eigenvalues of A . Then*

$$E \left[\sum_{i=1}^{n^2} \lambda_i^k \right] = E[\text{Trace}(A_F^k)]$$

$$\leq (2n^{1-2\gamma})^k + c \cdot k^4 \cdot k^{4k} \cdot 2^k \cdot (n^{(1-2\gamma)(k-1)} + n^2 \cdot n^{(1-2\gamma)k/2}),$$

where c is a constant (possibly $c = 100$ should be enough). If $k > 4/(1 - 2\gamma)$, the preceding estimate is $(2n^{1-2\gamma})^k + o((2n^{1-2\gamma})^k)$.

Proof. For any F we have that $\text{Trace}(A_F) = |\text{closed walks of length } k \text{ in } G_F|$. A typical closed walk of length k is

$$(a_0, b_0) \text{---} (a_1, b_1) \text{---} \dots \text{---} (a_{k-1}, b_{k-1}) \text{---} (a_k, b_k) = (a_0, b_0)$$

with the only constraint that adjacent vertices (= pairs of propositional variables) are different. Now consider a step $(a_{i-1}, b_{i-1}) \text{---} (a_i, b_i)$ of this walk. For this step to be possible in G_F the formula F must have one of the following $2n$ pairs of clauses: $a_{i-1} \vee a_i \vee z, b_{i-1} \vee b_i \vee \neg z$ for a propositional variable z or the other way around, that is, a_i, b_i first. We say that pairs of the first type induce the step $(a_{i-1}, b_{i-1}) \text{---} (a_i, b_i)$ with sign $+1$, whereas the second type induces this step with sign -1 . For two sequences of clauses $\vec{C} = (C_1, C_2, \dots, C_k)$, where the last literal of each C_i is a positive literal, and $\vec{D} = (D_1, D_2, \dots, D_k)$, where the last literal of each D_i is negative, and a sequence of signs $\vec{\varepsilon} = (\varepsilon_1, \dots, \varepsilon_k)$, we say that $\vec{C}, \vec{D}, \vec{\varepsilon}$ induce the walk above iff for each i the pair of clauses C_i, D_i induces the i th step of the walk with sign given by ε_i . Note that the occurrence of the clauses D_i and C_j in a random F is independent, as these clauses are always distinct. We say that F induces the walk above iff we can find sequences of clauses $\vec{C}, \vec{D} \subseteq F$ (the C_i, D_i need not necessarily be all distinct) and a sequence of signs $\vec{\varepsilon}$ such that $\vec{C}, \vec{D}, \vec{\varepsilon}$ induce the given walk. We observe that

- G_F allows for a given walk iff F induces this walk as defined above;
- three sequences $\vec{C}, \vec{D}, \vec{\varepsilon}$ induce at most one walk, but one walk can be induced by many $\vec{C}, \vec{D}, \vec{\varepsilon}$'s. (Without the $\vec{\varepsilon}$ it is possible that \vec{C}, \vec{D} induce several walks.)

Thus we get that $\text{Trace}(A_F^k)$ can be bounded above by the number of different sequences $\vec{C}, \vec{D}, \vec{\varepsilon}$ with $\vec{C}, \vec{D} \subseteq F$ inducing a closed walk of length k , and this estimate transfers to the expectation over a random formula F . The notions of distinct, quasi distinct, and duplicated are defined as for graphs: The sequence \vec{C} is *distinct* iff all component clauses C_i are different. \vec{C} is *quasi distinct* iff some C_i 's occur at least two times and some only once. \vec{C} is *duplicated* iff all C_i 's which occur in \vec{C} occur at least twice in \vec{C} . The same notions apply to \vec{D} . We decompose the expected number of $\vec{C}, \vec{D}, \vec{\varepsilon}$'s which induce a closed walk of length k according to all combinations of \vec{C}, \vec{D} being distinct, quasi distinct, or duplicated.

The expected number of $\vec{C}, \vec{D}, \vec{\varepsilon}$'s with \vec{C}, \vec{D} both distinct can be bounded as follows: The number of possible sequences \vec{C}, \vec{D} altogether can be bounded above

by n^{3k} . Note that we have unique correspondence between three sequences of variables $\vec{a} = (a_0, \dots, a_{k-1}, a_k = a_0)$, $\vec{b} = (b_0, \dots, b_{k-1}, b_k = b_0)$, and $\vec{z} = (z_1, \dots, z_k)$ and two sequences of clauses, which might induce a closed walk of length k , \vec{C}, \vec{D} : $C_i = a_{i-1} \vee a_i \vee z_i$ and $D_i = b_{i-1} \vee b_i \vee \neg z_i$. To account for the signs we get an additional factor of 2^k . The probability that $\vec{C}, \vec{D} \subseteq F$ for \vec{C}, \vec{D} distinct is $(1/n^{1+\gamma})^{2k}$ and we can bound the expectation by

$$2^k \cdot n^{3k} \left(\frac{1}{n^{1+\gamma}} \right)^{2k} = (2n^{1-2\gamma})^k.$$

Recall that $\gamma < 1/2$ and the degree of each vertex is concentrated at $2n^{1-2\gamma}$. Note also the analogous situation for $G_{n,p}$ above.

For \vec{C} distinct and \vec{D} duplicated we parameterize further with respect to the number l with $1 \leq l \leq k/2$ of different clauses D_i . Each \vec{D} possible at all is generated at least once as follows: 1. Pick l positions among the k possible positions where the different clauses D_i occur for the first time: $\binom{k}{l} \leq k^k$ possibilities. 2. For each of the $k-l$ remaining positions pick one of the preceding positions which were picked in 1: $\leq l^{k-l} \leq k^k$ possibilities. 3. Pick the clauses and the signs for the l positions from 1: $\leq n^{l+1} \cdot n^l \cdot 2^l$ possibilities. (Pick the b_i 's, pick the z_i 's, pick a sign.) 4. The remaining D_i 's are specified through 2 and 3, but perhaps we can pick a sign: 2^{k-j} possibilities. 5. Pick the sequence \vec{C} : n^k possibilities. (We can choose only the a_i 's.) For the expectation we get, observing that $l \leq k/2 < k$,

$$\begin{aligned} \sum_{l=1}^{k/2} k^{2k} \cdot 2^k \cdot n^{l+1} \cdot n^l \cdot n^k \cdot \left(\frac{1}{n^{1+\gamma}} \right)^{l+k} &\leq \sum k^{2k} \cdot 2^k \cdot n^{l+1} \cdot \left(\frac{1}{n^\gamma} \right)^{2l} \\ &= \sum_{l=1}^{k/2} k^{2k} \cdot 2^k \cdot n \cdot n^{(1-2\gamma)l} \leq k \cdot k^{2k} \cdot 2^k \cdot n \cdot n^{(1-2\gamma)k/2}, \end{aligned}$$

where we use $\gamma < 1/2$ and have omitted some of the factors $1/n^\gamma$ not necessary. Observe that $1 + (1 - 2\gamma)k/2 < (1 - 2\gamma)k$ iff $k > 2/(1 - 2\gamma)$. For this to hold k must become large when γ approaches $1/2$.

Now let \vec{C} be distinct and \vec{D} be quasi distinct. We first consider the case that the last clause from \vec{D} , D_k is unique. We parameterize further with respect to the number l with $1 \leq l \leq k - 2$ of different clauses in (D_1, \dots, D_{k-1}) . The number of possibilities for (D_1, \dots, D_{k-1}) altogether is bounded by $k^{2k} \cdot n^{l+1} \cdot n^l$. For D_1, \dots, D_{k-1}, D_k altogether we have only an additional factor of n . For the sign we get 2^k and for \vec{C} we have only at most n^k possibilities. Thus for the expectation we get, noting that always $l + 2 \leq k$,

$$\begin{aligned} \sum_{l=1}^{k-2} k^{2k} \cdot 2^k \cdot n^{l+1} \cdot n^l \cdot n \cdot n^k \cdot \left(\frac{1}{n^{1+\gamma}} \right)^{l+k} \cdot \frac{1}{n^{1+\gamma}} &\leq \sum k^{2k} \cdot 2^k \cdot n^{l+1} \cdot \left(\frac{1}{n^\gamma} \right)^{l+l+2} \\ &= \sum k^{2k} \cdot 2^k \cdot n^{(1-2\gamma)(l+1)} \leq k \cdot k^{2k} \cdot 2^k \cdot n^{(1-2\gamma)(k-1)}. \end{aligned}$$

Again we have omitted some unnecessary $1/n^\gamma$'s. To account for the fact that the unique D_i is in between we get an additional factor of k .

Now we come to the case that both \vec{C} and \vec{D} are duplicated. We parameterize further with respect to the number j ($1 \leq j \leq k/2$) of different C_i 's and l ($1 \leq l \leq k/2$)

of different D_i 's. Assume first $j \leq l$; the other case follows symmetrically. The number of different sequences $\vec{C}, \vec{D}, \vec{\varepsilon}$ altogether is bounded by $k^{4k} \cdot 2^k \cdot n^{j+1} \cdot n^j \cdot n^{l+1}$ as we choose the z_i 's with \vec{C} . Summing over j, l , this gives for the expectation a bound of

$$k^2 \cdot k^{4k} \cdot 2^k \cdot n^2 \cdot n^{(1-2\gamma)k/2}.$$

Note that $\gamma < 1/2$ and $2 + (1 - 2\gamma)k/2 < (1 - 2\gamma)k$ iff $k > 4/(1 - 2\gamma)$.

Now we look at the cases \vec{C} duplicated and \vec{D} quasi distinct. We first assume that the last clause of \vec{D} is unique. We parameterize further with respect to j , the number of distinct clauses in \vec{C} , and l , the number of distinct clauses in (D_1, \dots, D_{k-1}) . First assume that $j \leq l+1$. The number of possible sequences $\vec{C}, \vec{D}, \vec{\varepsilon}$ altogether is bounded above by $k^{4k} \cdot 2^k \cdot n^{j+1} \cdot n^j \cdot n^{l+1}$ as we choose the z_i 's with the j different C_i . For the expectation we get the sum going over $1 \leq j \leq k/2, 1 \leq l \leq k-2, j \leq l+1$

$$\begin{aligned} \sum k^{4k} \cdot 2^k \cdot n^{j+1} \cdot n^j \cdot n^{l+1} \cdot \left(\frac{1}{n^{1+\gamma}}\right)^{j+l} \cdot \frac{1}{n^{1+\gamma}} &\leq \sum k^{4k} \cdot 2^k \cdot n \cdot n^j \cdot \left(\frac{1}{n^\gamma}\right)^{2j} \\ &\leq k^2 \cdot k^{4k} \cdot 2^k \cdot n \cdot n^{(1-2\gamma)k/2}. \end{aligned}$$

Again note that $1 + (1 - 2\gamma)k/2 < (1 - 2\gamma)k$ for k a sufficiently large constant.

The case $l+1 \leq j$ yields the following in the same way, now choosing the z_i with the D_i 's:

$$\sum k^{4k} \cdot 2^k \cdot n^{l+1} \cdot n^l \cdot n \cdot n^{j+1} \cdot \left(\frac{1}{n^{1+\gamma}}\right)^{j+l} \cdot \frac{1}{n^{1+\gamma}} \leq k^2 \cdot k^{4k} \cdot 2^k \cdot n^2 \cdot n^{(1-2\gamma)k/2}.$$

For the unique clause of \vec{D} being somewhere in between we need an additional factor of k .

Now finally we look at the case that \vec{C}, \vec{D} are both quasi distinct. First assume that the two last clauses C_k, D_k are unique and let j be the number of different C_i among the first $k-1$ C_i 's and l the same for the D_i 's. First assume the $j \leq l$. We get for the expectation

$$\begin{aligned} \sum_{1 \leq j \leq l \leq k-2} k^{4k} \cdot 2^k \cdot n^{j+1} \cdot n^j \cdot n \cdot n^{l+1} \cdot \left(\frac{1}{n^{1+\gamma}}\right)^{l+j} \cdot \left(\frac{1}{n^{1+\gamma}}\right)^2 \\ \leq k^2 \cdot k^{4k} \cdot 2^k \cdot n^{(1-2\gamma)(k-1)}. \end{aligned}$$

For the unique clauses standing at different positions we get the same estimate, which altogether accounts for another factor of k^2 . For $l \leq j$ we get another factor of 2. \square

Now our algorithm is obvious: We pick ε, ν sufficiently small such that the $f(\nu, \varepsilon)$ from Theorem 12 is $< 1/20$. Given $F \in \text{Form}_{n,p}, p = 1/n^{1+\gamma}$, we construct G_F . We check if maximum degree/minimum degree $\leq (1 + \varepsilon)/(1 - \varepsilon)$. This holds with high probability; when it does not, the algorithm fails. Now we determine λ_1 and λ . With high probability we have that $\lambda \leq \nu\lambda_1$. If the last estimate does not hold, we fail. By Theorem 12 the algorithm now has certified that G_F has no independent set of size $\geq n^2/4$. We do the same for H_F . With high probability we succeed and by Theorem 8 F is certified unsatisfiable.

Conclusion. Our algorithm works with high probability with respect to the binomial space $\text{Form}_{n,p}$, where p is such that the *expected number* of clauses is the announced $n^{3/2+\varepsilon}$. When we always draw *exactly* $n^{3/2+\varepsilon}$ clauses the algorithm should

also work, as can be shown by applying very similar arguments to those given in the paper. Note that we generate formulas in $\text{Form}_{n,p}$ with exactly the expectation of many clauses with probability bounded below only $\Omega(1/\sqrt{n})$. However, after generating exactly $n^{3/2+\varepsilon}$ clauses one can formally delete some clauses with the appropriate low probability. This yields a subset having high probability on $\text{Form}_{n,p}$ with an appropriate p and the preceding consideration applies to these new formulas, thus showing the unsatisfiability of the original formula.

REFERENCES

- [Ac2000] D. ACHLIOPTAS, *Setting 2 variables at a time yields a new lower bound for random 3-SAT*, in Proceedings of the 32nd Annual Symposium on Theory of Computing, ACM, New York, 2000, pp. 28–37.
- [AcFr99] D. ACHLIOPTAS AND E. FRIEDGUT, *A sharp threshold for k -colourability*, Random Structures Algorithms, 14 (1999), pp. 63–70.
- [AcMo97] D. ACHLIOPTAS AND M. MOLLOY, *The analysis of a list-colouring algorithm on a random graph*, in Proceedings of the 38th Annual Symposium on Foundations of Computer Science, IEEE, Los Alamitos, CA, 1997, pp. 204–212.
- [AcMo99] D. ACHLIOPTAS AND M. MOLLOY, *Almost all graphs with $2.522n$ edges are not 3-colourable*, Electron. J. Combin., 6 (1999), Research paper 29.
- [AcMo2002] D. ACHLIOPTAS AND C. MOORE, *The asymptotic order of the random k -SAT threshold*, in Proceedings of the 43rd Annual Symposium on Foundations of Computer Science, IEEE, Los Alamitos, CA, 2002, pp. 779–788.
- [AcPe2004] D. ACHLIOPTAS AND Y. PERES, *The threshold for random k -SAT is $2^k(\log 2 - o(1))$* , J. Amer. Math. Soc., 17 (2004), pp. 947–973.
- [AcSo2000] D. ACHLIOPTAS AND G. B. SORKIN, *Optimal myopic algorithms for random 3-SAT*, in Proceedings of the 41st Annual Symposium on Foundations of Computer Science, IEEE, Los Alamitos, CA, 2000, pp. 590–600.
- [Al98] N. ALON, *Spectral techniques in graph algorithms*, in Proceedings of LATIN 1998, Lecture Notes in Comput. Sci. 1380, Springer, Berlin, 1998, pp. 206–215.
- [AlKa97] N. ALON AND N. KAHALE, *A spectral technique for coloring random 3-colorable graphs*, SIAM J. Comput., 26 (1997), pp. 1733–1748.
- [AlSp92] N. ALON AND J. H. SPENCER, *The Probabilistic Method*, John Wiley & Sons, New York, 1992.
- [Be at al2002] P. BEAME, R. KARP, T. PITASSI, AND M. SAKS, *The efficiency of resolution and Davis–Putnam procedures*, SIAM J. Comput., 31 (2002), pp. 1048–1075.
- [BePi96] P. BEAME AND T. PITASSI, *Simplified and improved resolution lower bounds*, in Proceedings of the 37th Annual Symposium on Foundations of Computer Science, IEEE, Los Alamitos, CA, 1996, pp. 274–282.
- [Bl et al81] M. BLUM, R. KARP, O. VORNBERGER, C. H. PAPADIMITRIOU, AND M. YANNAKAKIS, *The complexity of testing whether a graph is a superconcentrator*, Inform. Process. Lett., 13 (1981), pp. 164–167.
- [Bo85] B. BOLLOBÁS, *Random Graphs*, Academic Press, New York, 1985.
- [BoChPi2001] C. BORGS, J. CHAYES, AND B. PITTEL, *Phase transition and finite-size scaling for the integer partitioning problem*, Random Structures Algorithms, 19 (2001), pp. 247–288.
- [Ch97] F. R. K. CHUNG, *Spectral Graph Theory*, AMS, Providence, RI, 1997.
- [ChRe92] V. CHVATAL AND B. REED, *Mick gets some (the odds are on his side)*, in Proceedings of the 33rd Annual Symposium on Foundations of Computer Science, IEEE, Los Alamitos, CA, 1992, pp. 620–627.
- [ChSz88] V. CHVATAL AND E. SZEMERÉDI, *Many hard examples for resolution*, J. ACM, 35 (1988), pp. 759–768.
- [CoGoLa 2004] A. COJA-OGHLAN, A. GOERDT, AND A. LANKA, *Strong refutation heuristics for random k -SAT*, in Proceedings of the 8th International Workshop on Randomization and Computation, Lecture Notes in Comput. Sci. 3122, Springer, Berlin, 2004, pp. 310–322.

- [CoGoLaSch 2004] A. COJA-OGLHAN, A. GOERDT, A. LANKA, AND F. SCHÄDLICH, *Techniques from combinatorial approximation algorithms yield efficient algorithms for random $2k$ -SAT*, Theoret. Comput. Sci., 329 (2004), pp. 1–45.
- [CrAu96] J. M. CRAWFORD AND L. D. AUTON, *Experimental results on the crossover point in random 3-SAT*, Artificial Intelligence, 81 (1996), pp. 31–57.
- [DuBoMa2000] O. DUBOIS, Y. BOUFGHAD, AND J. MANDLER, *Typical random 3-SAT formulae and the satisfiability threshold*, in Proceedings of the 11th Annual ACM-SIAM Symposium on Discrete Algorithms, SIAM, Philadelphia, 2000, pp. 126–127.
- [DuZi98] P. E. DUNNE AND M. ZITO, *An improved upper bound for the non-3-colourability threshold*, Inform. Process. Lett., 65 (1998), pp. 17–23.
- [FeOf 2004] U. FEIGE AND E. OFEK, *Easily refutable subformulas of large random 3-CNF formulas*, in Proceedings of the 31st International Colloquium on Automata, Languages and Programming (ICALP), Lecture Notes in Comput. Sci. 3142, Springer, Berlin, 2004, pp. 519–530.
- [Fr91] J. FRIEDMAN, *On the second eigenvalue and random walks in random d -regular graphs*, Combinatorica, 11 (1991), pp. 331–362.
- [Fr99] E. FRIEDGUT, *Necessary and sufficient conditions for sharp thresholds of graph properties and the k -SAT problem*, J. Amer. Math. Soc., 12 (1999), pp. 1017–1054.
- [FrSu96] A. M. FRIEZE AND S. SUEN, *Analysis of two simple heuristics on a random instance of k -SAT*, J. Algorithms, 20 (1996), pp. 312–355.
- [Fu95] X. FU, *On the Complexity of Proof Systems*, Ph.D. thesis, University of Toronto, Toronto, Ontario, Canada, 1995.
- [FuKo81] Z. FÜREDI AND J. KOMLÓS, *The eigenvalues of random symmetric matrices*, Combinatorica, 1 (1981), pp. 233–241.
- [Go96] A. GOERDT, *A threshold for unsatisfiability*, J. Comput. System Sci., 53 (1996), pp. 469–486.
- [GoKr2001] A. GOERDT AND M. KRIVELEVICH, *Efficient recognition of random unsatisfiable k -SAT instances by spectral methods*, in Proceedings of STACS 2001, Lecture Notes in Comput. Sci. 2010, Springer, Berlin, 2001, pp. 294–304.
- [ImNa96] R. IMPAGLIAZZO AND M. NAOR, *Efficient cryptographic schemes provably as secure as subset sum*, J. Cryptology, 9 (1996), pp. 199–216.
- [Ju82] F. JUHÁSZ, *The asymptotic behaviour of Lovász theta function for random graphs*, Combinatorica, 2 (1982), pp. 153–155.
- [KaKiLa2002] A. C. KAPORIS, L. M. KIROUSIS, AND E. G. LALAS, *The probabilistic analysis of a greedy satisfiability algorithm*, in Proceedings of the 10th ESA, Lecture Notes in Comput. Sci. 2461, Springer, Berlin, 2002, pp. 574–585.
- [KiKrKr98] L. M. KIROUSIS, E. KRANAKIS, D. KRIZANC, AND Y. STAMATIOU, *Approximating the unsatisfiability threshold of random formulas*, Random Structures Algorithms, 12 (1998), pp. 253–269.
- [KrVu2002] M. KRIVELEVICH AND V. H. VU, *Approximating the independence number and the chromatic number in expected polynomial time*, J. Comb. Optim., 6 (2002), pp. 143–155.
- [PeWe89] A. D. PETFORD AND D. WELSH, *A randomised 3-colouring algorithm*, Discrete Math., 74 (1989), pp. 253–261.
- [Sch89] U. SCHÖNING, *Logic for Computer Scientists*, Birkhäuser, Boston, 1989.
- [SeMiLe96] B. SELMAN, D. G. MITCHELL, AND H. J. LEVESQUE, *Generating hard satisfiability problems*, Artificial Intelligence, 81 (1996), pp. 17–29.
- [St88] G. STRANG, *Linear Algebra and Its Applications*, Harcourt Brace Jovanovich, San Diego, CA, 1988.

OPTIMAL ONLINE ALGORITHMS FOR MULTIDIMENSIONAL PACKING PROBLEMS*

LEAH EPSTEIN[†] AND ROB VAN STEE[‡]

Abstract. We solve an open problem in the literature by providing an online algorithm for multidimensional bin packing that uses only bounded space. To achieve this, we introduce a new technique for classifying the items to be packed. We show that our algorithm is optimal among bounded space algorithms for any dimension $d > 1$. Its asymptotic performance ratio is $(\Pi_\infty)^d$, where $\Pi_\infty \approx 1.691$ is the asymptotic performance ratio of the one-dimensional algorithm HARMONIC. A modified version of this algorithm for the case where all items are hypercubes is also shown to be optimal. Its asymptotic performance ratio is sublinear in d .

Furthermore, we extend the techniques used in these algorithms to give optimal algorithms for online bounded space variable-sized packing and resource augmented packing.

Key words. multidimensional bin packing, online algorithms, optimal algorithms

AMS subject classifications. 68Q25, 68W40

DOI. 10.1137/S0097539705446895

1. Introduction. Bin packing is one of the oldest and most well-studied problems in computer science [12, 6]. The study of this problem dates back to the early 1970's, when computer science was still in its formative phase—ideas which originated in the study of the bin packing problem have helped shape computer science as we know it today. The influence and importance of this problem are witnessed by the fact that it has spawned off whole areas of research, including the fields of online algorithms and approximation algorithms. In this paper, we study a natural generalization of bin packing called box packing.

Problem definition: Let $d \geq 1$ be an integer. In the d -dimensional box packing problem, we receive a sequence σ of items h_1, h_2, \dots, h_n . Each item h has a fixed size, which is $s_1(h) \times \dots \times s_d(h)$, i.e., $s_i(h)$ is the size of h in the i th dimension. We have an infinite number of bins, each of which is a d -dimensional unit hypercube. Each item must be assigned to a bin and a position $(x_1(h), \dots, x_d(h))$, where $0 \leq x_i(h)$ and $x_i(h) + s_i(h) \leq 1$ for $1 \leq i \leq d$. Further, the positions must be assigned in such a way that no two items in the same bin overlap. A bin is *empty* if no item is assigned to it, otherwise it is *used*. The goal is to minimize the number of bins used. Note that for $d = 1$, the box packing problem reduces to exactly the classic bin packing problem.

There are a number of variants of this problem which are of interest:

- In the *online* version of this problem, each item must be assigned in turn, without knowledge of the next items.

*Received by the editors January 24, 2005; accepted for publication (in revised form) July 5, 2005; published electronically November 9, 2005. Preliminary versions of different parts of this paper appeared in *Proceedings of the Symposium of Discrete Algorithms 2004* (SODA 2004) and *Proceedings of the European Symposium Algorithm 2004* (ESA 2004).

<http://www.siam.org/journals/sicomp/35-2/44689.html>

[†]Department of Mathematics, University of Haifa, 31905 Haifa, Israel (lea@math.haifa.ac.il). This author's research was supported by the Israel Science Foundation (grant 250/01).

[‡]Fakultät für Informatik, Universität Karlsruhe, D-76128 Karlsruhe, Germany (vanstee@ira.uka.de). This author's research was supported by the Netherlands Organization for Scientific Research (NWO), project SION 612-061-000. This work was performed while this author was at CWI, The Netherlands.

- In the *hypercube packing* problem we have the restriction that all items are hypercubes, i.e., an item has the same size in every dimension.
- In the *bounded space* variant, an algorithm has only a constant number of bins available to accept items at any point during processing. The bounded space assumption is a quite natural one, especially so in online box packing. Essentially the bounded space restriction guarantees that output of packed bins is steady, and that the packer does not accumulate an enormous backlog of bins which are only output at the end of processing.
- In *variable-sized* bin packing, bins of various sizes are available to be used for packing and the goal is to minimize the total size of all the bins used.
- In *resource-augmented* bin packing, the online algorithm has larger bins at its disposal than the offline algorithm, and the goal is to minimize the number of bins used.

The offline versions of these problems are NP-hard, while even with unlimited computational ability it is impossible in general to produce the best possible solution online. We consider online approximation algorithms.

The standard measure of algorithm quality for box packing is the *asymptotic performance ratio*, which we now define. For a given input sequence σ , let $\text{cost}_{\mathcal{A}}(\sigma)$ be the number of bins used by algorithm \mathcal{A} on σ . Let $\text{cost}(\sigma)$ be the minimum possible number of bins used to pack items in σ . The *asymptotic performance ratio* for an algorithm \mathcal{A} is defined to be

$$\mathcal{R}_{\mathcal{A}}^{\infty} = \limsup_{n \rightarrow \infty} \sup_{\sigma} \left\{ \frac{\text{cost}_{\mathcal{A}}(\sigma)}{\text{cost}(\sigma)} \mid \text{cost}(\sigma) = n \right\}.$$

Let \mathcal{O} be some class of box packing algorithms (for instance, online algorithms or bounded space online algorithms). The *optimal asymptotic performance ratio* for \mathcal{O} is defined to be $\mathcal{R}_{\mathcal{O}}^{\infty} = \inf_{\mathcal{A} \in \mathcal{O}} \mathcal{R}_{\mathcal{A}}^{\infty}$. Given \mathcal{O} , our goal is to find an algorithm with an asymptotic performance ratio close to $\mathcal{R}_{\mathcal{O}}^{\infty}$.

Previous results: The classic (one-dimensional) online bin packing problem was first investigated by Ullman [33]. He showed that the FIRST FIT algorithm has performance ratio $\frac{17}{10}$. This result was then published in [20]. Johnson [22] showed that the NEXT FIT algorithm has a performance ratio of 2. Yao showed that REVISED FIRST FIT has a performance ratio of $\frac{5}{3}$, and further showed that no online algorithm has a performance ratio of less than $\frac{3}{2}$ [38]. Brown and Liang independently improved this lower bound to 1.53635 [3, 28]. The lower bound currently stands at 1.54014, due to van Vliet [34]. Define

$$\pi_{i+1} = \pi_i(\pi_i - 1) + 1, \quad \pi_1 = 2,$$

and

$$\Pi_{\infty} = \sum_{i=1}^{\infty} \frac{1}{\pi_i - 1} \approx 1.69103.$$

Lee and Lee presented an algorithm called HARMONIC, which uses $m > 1$ classes and uses bounded space. For any $\varepsilon > 0$, there is a number m such that the HARMONIC algorithm that uses m classes has a performance ratio of at most $(1 + \varepsilon)\Pi_{\infty}$ [25]. They also showed there is no bounded space algorithm with a performance ratio below Π_{∞} . Bounded space algorithms for bin packing were also considered by Woeginger in [37]

and by Van Vliet in [36]. Currently the best known unbounded space upper bound is 1.58889 due to Seiden [31].

Offline bin packing has also received a great deal of attention, for a survey see [6]. The most prominent results are as follows: Garey, Graham, and Ullman [20] were the first to study the approximation ratios of both online and offline algorithms. Fernandez de La Vega and Lueker [15] presented the first (asymptotic) approximation scheme for bin packing. Karmarkar and Karp [23] gave an algorithm which uses at most $\text{cost}(\sigma) + \log^2(\text{cost}(\sigma))$ bins.

The online one-dimensional variable-sized bin packing problem was first investigated by Friesen and Langston [17]. Csirik [9] proposed the VARIABLE HARMONIC algorithm and showed that it has performance ratio at most Π_∞ . Seiden [30] showed that this algorithm is optimal among bounded space algorithms.

The online one-dimensional resource augmented bin packing problem was studied by Csirik and Woeginger [13]. They showed that the optimal bounded space asymptotic performance ratio is a function $\rho(b)$ of the size b of the bins of the online algorithm.

While box packing is a natural next step from bin packing, the problem seems to be more difficult, and the number of results is smaller. The offline problem was introduced by Chung, Garey, and Johnson [5]. Caprara [4] presented an algorithm with an approximation ratio of Π_∞ for $d = 2$.

The online problem was first investigated by Coppersmith and Raghavan [7], who give an algorithm based on NEXT FIT with performance ratio $\frac{13}{4} = 3.25$ for $d = 2$. Csirik, Frenk, and Labbe [10] gave an algorithm based on FIRST FIT with a performance ratio of $\frac{49}{16} = 3.0625$ for $d = 2$. Csirik and van Vliet [11] presented an algorithm with a performance ratio of $(\Pi_\infty)^d$ for all $d \geq 2$ (2.85958 for $d = 2$). Even though this algorithm is based on HARMONIC, it was not clear how to change it to bounded space. Li and Cheng [27] also gave a HARMONIC-based algorithm for $d = 2$ and $d = 3$.

Seiden and van Stee [32] improved the upper bound for $d = 2$ to 2.66013. Several lower bounds have been shown [18, 19, 35, 2]. The best lower bound for $d = 2$ is 1.907 [2], while the best lower bound for large d is less than 3. For bounded space algorithms, a lower bound of $(\Pi_\infty)^d$ is implied by [11].

For online square packing, even less is known. The following results are known for $d = 2$: Coppersmith and Raghavan [7] showed an upper bound of $43/16 = 2.6875$ and a lower bound of $4/3$ (which holds for all $d \geq 2$). The upper bound was improved to $395/162 < 2.43828$ by Seiden and van Stee [32]. For $d = 3$, Miyazawa and Wakabayashi [29] showed an upper bound of 3.954. For the offline problem, Ferreira, Miyazawa, and Wakabayashi give a 1.988-approximation algorithm [16]. A sequence of results improved this result [24, 32, 4], recently culminating in an APTAS by Bansal and Sviridenko [1] and Correa and Kenyon [8] independently.

Our results: In this paper, we present a number of results for online and offline box and square packing:

- We begin by presenting a bounded space algorithm for the packing of hypercubes. An interesting feature of the analysis is that although we show the algorithm is optimal, we do not know the exact asymptotic performance ratio; the asymptotic performance ratio is $\Omega(\log d)$ and $O(d/\log d)$.
- We then extend this algorithm to a bounded space algorithm for general hyperbox packing and show that this algorithm is also optimal, with an asymptotic performance ratio of $(\Pi_\infty)^d$. This solves the ten-year-old open problem

of how to pack hyperboxes using only bounded space.

- We present a bounded space algorithm for the variable-sized multidimensional bin packing problem. As for the first algorithm above, we do not know the exact asymptotic performance ratio.
- We then give an analogous algorithm for the problem of resource augmented online bin packing. This algorithm is also optimal, with an asymptotic performance ratio of $\prod_{i=1}^d \rho(b_i)$ where $b_1 \times \dots \times b_d$ is the size of the bins that the online algorithm uses.

For the online results, we will use the technique of weighting functions. This technique was originally introduced for one-dimensional bin packing algorithms [33, 21]. In [32], it was demonstrated how to use the analysis for one-dimensional algorithms to get results for higher dimensions. In contrast, in the current paper we will define weighting functions directly for multidimensional algorithms, without using one-dimensional algorithms as subroutines.

New technique: To construct the bounded space algorithm we adapt some of the ideas used in previous work. Specifically, the algorithm of [11] also required a scheme of partitioning bins into sub-bins, and of sub-bins into smaller and smaller sub-bins. However, in order to keep a constant number of bins active, we had to introduce a new method of classifying items. Our key improvement is that there is not one single class of “small” items like all the standard algorithms have, but instead we partition the items into an infinite number of classes that are grouped into a finite number of groups. The hypercube packing algorithm uses an easier scheme for the same purpose. This is a more direct extension of the method used in [7].

1.1. The harmonic algorithm. In this section we briefly discuss the important one-dimensional HARMONIC algorithm [25]. In the next sections we adapt it to the multidimensional cases using our novel techniques.

The fundamental idea of these algorithms is to first classify items by size, and then pack an item according to its class (as opposed to letting the exact size influence packing decisions).

For the classification of items, we need to partition the interval $(0, 1]$ into subintervals. The standard HARMONIC algorithm uses $M - 1$ subintervals of the form $(1/(i + 1), 1/i]$ for $i = 1, \dots, M - 1$ and one final subinterval $(0, 1/M]$. Each bin will contain only items from one subinterval (type). Items in subinterval i are packed i per bin for $i = 1, \dots, M - 1$ and the items in interval M are packed in bins using NEXT FIT (i.e., a greedy algorithm that opens a new active bin whenever an item does not fit into the current active bin and never uses the previous bins).

2. Packing hypercubes. In this section we define the algorithm for hypercubes denoted by ALG_ε . In the next section we extend the algorithm to deal with hyperboxes. Let the *size* of hypercube h , $s(h)$ be the length of each side of the hypercube.

The algorithm has a parameter $\varepsilon > 0$. Let $M \geq 10$ be an integer parameter such that $M \geq 1/(1 - (1 - \varepsilon)^{1/(d+1)}) - 1$. We distinguish between “small” hypercubes (of size smaller or equal to $1/M$) and “big” hypercubes (of size larger than $1/M$). The packing algorithm will treat them in different ways.

All *large* hypercubes are packed using a multidimensional version of HARMONIC [25]. The hypercubes are assigned a type according to their size: type i items have a size in the interval $(1/(i + 1), 1/i]$ for $i = 1, \dots, M - 1$. The bins that are used to pack items of these types all contain items of only one type. We use the following algorithm to pack them. A bin is called *active* if it can still receive items, otherwise it is *closed*.

Algorithm ASSIGNLARGE(i). At all times, there is at most one active bin for each type. Each bin is partitioned into i^d hypercubes (sub-bins) of size $1/i$ each (the sub-bins create a grid of i strips in each dimension). Each such sub-bin can contain exactly one item of type i . On arrival of a type i item it is assigned to a free sub-bin (and placed anywhere inside this sub-bin). If all sub-bins are taken, the previous active bin is closed, a new active bin is opened and partitioned into sub-bins.

The *small* hypercubes are also assigned types depending on their size, but in a different way. Consider an item h of size $s(h) \leq 1/M$. Let k be the largest nonnegative integer such that $2^k s(h) \leq 1/M$; clearly $2^k s(h) > 1/(2M)$. Let i be the integer such that $2^k s(h) \in (1/(i+1), 1/i]$, $i \in \{M, \dots, 2M-1\}$. The item is defined to be of type i . Each bin that is used to pack small items contains only small items with a given type i . Note that items of very different sizes may be packed together in one bin. We now describe the algorithm to pack a new small item of type i for $i = M, \dots, 2M-1$. A sub-bin which received a hypercube is said to be *used*. A sub-bin which is not used and not cut into smaller sub-bins is called *empty*.

Algorithm ASSIGNSMALL(i). The algorithm maintains a single active bin. Each bin may, during its use, be partitioned into sub-bins which are hypercubes of different sizes of the form $1/(2^j i)$. When an item h of type i arrives we perform the following. Let k be the integer such that $2^k s(h) \in (1/(i+1), 1/i]$.

1. If there is an empty sub-bin of size $1/(2^k i)$, then the item is simply assigned there and placed anywhere within the sub-bin.
2. Else, if there is no empty sub-bin of any size $1/(2^j i)$ for $j < k$ inside the current bin, the bin is closed and a new bin is opened and partitioned into sub-bins of size $1/i$. Then the procedure in step 3 is followed, or step 1 in case $k = 0$.
3. Take an empty sub-bin of size $1/(2^j i)$ for a maximum $j < k$. Partition it into 2^d identical sub-bins (by cutting into two identical pieces, in each dimension). If the resulting sub-bins are larger than $1/(2^k i)$, take *one* of them and partition it in the same way. This is done until sub-bins of size $1/(2^k i)$ are reached. The new item is assigned into one such sub-bin.

Finally, the main algorithm only determines the type of newly arriving items and assigns them to the appropriate algorithms. The total number of active bins is at most $2M-1$. In order to perform a competitive analysis, we prove the following claims.

CLAIM 1. *For a given $i \geq M$, consider an active bin of type i . At all times, the number of empty sub-bins in it of each size except $1/i$ is at most $2^d - 1$.*

Proof. Note that the number of empty sub-bins of size $1/i$ decays from i^d to zero during the usage of such a bin. Consider a certain possible size r of a sub-bin in it. When a sub-bin of some size r is created, it is due to a partition of a larger sub-bin. This means that there were no empty sub-bins of size r before the partition. Afterwards, there are at most $2^d - 1$ of them for each size that has been created during the partitioning (for the smallest size into which the sub-bin is partitioned, 2^d sub-bins created, but one is immediately used). \square

CLAIM 2. *For a given $i \geq M$, when a bin of type i is about to be closed, the total volume of empty sub-bins in the bin is at most $1/i^d$.*

Note that the above claims bound the volume of sub-bins that are not used at all. There is some waste of volume also due to the fact that each item does not fill its sub-bin totally. We compute this waste later.

Proof. For $i \geq M$, when a bin of type i is to be closed, there are no empty sub-bins of size $1/i$ in it. There are at most $2^d - 1$ empty sub-bins of each other size by Claim 1. This gives a total unused volume of at most $(2^d - 1) \sum_{k \geq 1} (2^k i)^{-d} = 1/i^d$. \square

CLAIM 3. *The occupied volume in each closed bin of type $i \geq M$ is at least $1 - \varepsilon$.*

Proof. A hypercube which was assigned into a sub-bin of size $1/(2^k i)$ always has size of at least $1/(2^k(i+1))$. Therefore the ratio of occupied space and existing space in each used sub-bin is at least $i^d/(i+1)^d$. When a bin is closed, the total volume of used sub-bins is at least $1 - 1/i^d$ by Claim 2. Therefore the occupied volume in the bin is at least $i^d/(i+1)^d(1 - 1/i^d) = (i^d - 1)/(i+1)^d$. We use $i \geq M$ and $M^d \geq M + 1$ to get $(i^d - 1)/(i+1)^d \geq (M^d - 1)/(M+1)^d \geq (\frac{M}{M+1})^{d+1} \geq 1 - \varepsilon$. \square

Now we are ready to analyze the performance. We define a weighting function for ALG_ε [33]. Each item p with type $1 \leq i \leq M - 1$ has weight $w_\varepsilon(p) = 1/i^d$. Each item p' of higher type has weight $w_\varepsilon(p') = (s(p))^d/(1 - \varepsilon)$ which is the volume of the item divided by $(1 - \varepsilon)$. We begin by showing that this weighting function is valid for our algorithm.

LEMMA 2.1. *For all input sequences σ , $\text{cost}_{\text{ALG}_\varepsilon}(\sigma) \leq \sum_{h \in \sigma} w_\varepsilon(h) + 2M - 1$.*

Proof. Each closed bin of type $1 \leq i \leq M - 1$ contains i^d items. All sub-bins are used when the bin is closed, and thus it contains a total weight of 1. Each closed bin of type $M \leq i \leq 2M - 1$ has an occupied volume of at least $1 - \varepsilon$ by Claim 3, and therefore the weights of the items in such a bin sum up to at least 1. At most $2M - 1$ bins are active. Thus the total number of bins used by ALG_ε for a given input sequence σ is upper bounded by the total weight of the items plus $2M - 1$. \square

By Lemma 2.1, for any given $\varepsilon > 0$, the asymptotic performance ratio of our algorithm can be upper bounded by the maximum amount of weight that can be packed in a single bin: for a given input sequence σ (with fixed weight w), the offline algorithm minimizes the number of bins that it needs to pack all items in σ by packing as much weight as possible in each bin. If it needs k bins, the performance ratio on this input is w/k , which is also the average weight per offline bin.

Therefore we need to find the worst case offline bin, i.e., an offline bin which is packed with a maximum amount of weight. However, for the case of cubes, we only have $M + 1$ different types of items. All large items of type i have the same weight. All small items have the same ratio of weight to volume. Therefore the exact contents of a bin are not crucial. In order to define a packed bin, we need only to know how many items there are of each type, and the volume of the small items. To maximize the weight we can assume that the large items are as small as possible (without changing their type), and the rest of the bin is filled with small items.

Formally, we define a *pattern* as a tuple $q = \langle q_1, \dots, q_{M-1} \rangle$, where there exists a feasible packing into a single bin containing q_i items of type i for all $1 \leq i \leq M - 1$. This generalizes the definition from [31]. The weight of a pattern q is at most

$$(1) \quad w_\varepsilon(q) = \sum_{i=1}^{M-1} \frac{q_i}{i^d} + \frac{1}{1 - \varepsilon} \left(1 - \sum_{i=1}^{M-1} \frac{q_i}{(i+1)^d} \right).$$

Note that for any given pattern the amounts of items of types $M, \dots, 2M - 1$ are unspecified. However, as mentioned above, the weight of such items is always their volume divided by $1 - \varepsilon$. Therefore (1) gives an upper bound for the total weight that can be packed in a single bin for a given pattern q . Summarizing, we have the following theorem.

THEOREM 2.2. *The asymptotic performance ratio of ALG_ε is upper bounded by $\max_q w_\varepsilon(q)$, where the maximum is taken over all patterns q that are valid for ALG_ε .*

In order to use Theorem 2.2, we need the following geometric claim. We immediately formulate it in a general way so that we can also apply it in the next section.

CLAIM 4. *Given a packing of hyperboxes into bins, such that component j of each hyperbox is bounded in an interval $(1/(k_j + 1), 1/k_j]$, where $k_j \geq 1$ is an integer for $j = 1, \dots, d$, then each bin has at most $\prod_{j=1}^d k_j$ hyperboxes packed in it.*

Proof. We prove the claim by induction on the dimension. Clearly for $d = 1$ the claim holds. To prove the claim for $d > 1$, the induction hypothesis means that a hyperplane of dimension $d - 1$ through the bin which is parallel to one of the sides (the side which is the projection of the bin on the first $d - 1$ dimensions) can meet at most $\prod_{j=1}^{d-1} k_j$ hyperboxes. Next, take the projection of the hyperboxes and the bin on the last axis. We get short intervals of length in $(1/(k_d + 1), 1/k_d]$ (projections of hypercubes) on a main interval of length 1 (the projection of the bin). As mentioned above, each point of the main interval can have the projection of at most $\prod_{j=1}^{d-1} k_j$ items. Consider the short intervals as an interval graph. The size of the largest clique is at most $\prod_{j=1}^{d-1} k_j$. Therefore, as interval graphs are perfect, we can color the short intervals using $\prod_{j=1}^{d-1} k_j$ colors. Note that the number of intervals of each independent set is at most k_d (due to length), and so the total number of intervals is at most $\prod_{j=1}^d k_j$. \square

LEMMA 2.3. *Let $\alpha = \liminf_{\varepsilon \rightarrow 0} \max_q w_\varepsilon(q)$, where the maximum is taken over all patterns q that are valid for ALG_ε . Then the asymptotic performance ratio of any bounded space algorithm is at least α .*

Proof. We show that there is no bounded space algorithm with an asymptotic performance ratio strictly below α . For any $\varepsilon' > 0$, there exists an $\varepsilon \in (0, \varepsilon')$ such that $\mathcal{R}^\infty(\text{ALG}_\varepsilon) \leq (1 + \varepsilon')\alpha$. Consider the pattern q for which $w_\varepsilon(q)$ is maximal. We write $w_\varepsilon(q) = (1 + \varepsilon'')\alpha$ for some $\varepsilon'' \in [0, \varepsilon']$.

Note that a pattern does not specify the precise sizes of any of the items in it. Based on q , we define a set of hypercubes that can be packed together in a single bin. For each item of type i in q , we take a hypercube of size $1/(i + 1) + \delta$ for some small $\delta > 0$. Take $V_\delta = 1 - \sum_{i=1}^{M-1} q_i(1/(i + 1) + \delta)^d$. We add a large amount of small hypercubes of total volume V_δ , where the sizes of the small hypercubes are chosen in such a way that they can all be packed in a single bin together with the large hypercubes prescribed by q . By the definition of a pattern, such a packing is feasible for δ sufficiently small.

Define the following input for a bounded space algorithm. Let N be a large constant. The sequence contains M phases. The last phase contains a volume NV_δ of small hypercubes. Phase i ($1 \leq i \leq M - 1$) contains Nq_i hypercubes of size $1/(i + 1) + \delta$. After phase i , almost all hypercubes of this phase must be packed into closed bins (except a constant number of active bins). Each such bin may contain up to i^d items, which implies that in each phase i , $Nq_i/i^d - O(1)$ bins are closed. The last phase contributes at least $V_\delta - O(1)$ extra bins. The cost of the online algorithm is $\sum_{i=1}^{M-1} Nq_i/i^d + V_\delta - O(M)$. But the optimal offline cost is simply N . Taking $\delta = 1/N$ and letting N grow without bound, N becomes much larger than M and the asymptotic performance ratio of any bounded space online algorithm is lower bounded by $\sum_{i=1}^{M-1} q_i/i^d + V_0$. Note that the weight of this set of hypercubes according to our definition of weights tends to $\sum_{i=1}^{M-1} q_i/i^d + V_0/(1 - \varepsilon) = w_\varepsilon(q) = (1 + \varepsilon'')\alpha$ as $\delta \rightarrow 0$. Therefore $\sum_{i=1}^{M-1} q_i/i^d + V_0 \geq (1 - \varepsilon)(1 + \varepsilon'')\alpha \geq (1 - \varepsilon')\alpha$. \square

Lemma 2.3 implies that our algorithm is the best possible bounded space algorithm. More precisely, for every $\varepsilon' > 0$, there exists an $\varepsilon \in (0, \varepsilon')$ such that

$\mathcal{R}^\infty(\text{ALG}_\varepsilon) \leq (1 + \varepsilon')\alpha$, and no bounded space algorithm has an asymptotic performance ratio below $(1 - \varepsilon')\alpha$. This also implies that our weighting function cannot be improved and determines the asymptotic performance ratio exactly. However, we have no general formula for this ratio. We do have the following bounds.

THEOREM 2.4. *There exists a value of M such that the asymptotic performance ratio of ALG_ε is $O(d/\log d)$. Any bounded space algorithm (in particular ALG_ε) has an asymptotic performance ratio of $\Omega(\log d)$.*

Proof. We first show the upper bound. Take $M = 2d/\log d$. The occupied area in bins of small types is at least $(\frac{M}{M+1})^{d+1}$ by the proof of Claim 3. This is greater than $(\frac{M+1}{M})^{-d} = (1 + 1/M)^{-d} = (1 + (\log d)/(2d))^{-d}$, which tends to $e^{-(\log d)/2} = (e^{\log d})^{-1/2} = 1/\sqrt{d}$ for $d \rightarrow \infty$.

Suppose the input is I . Denote by I_i the subsequence of items of type i ($i = 1, \dots, M$), where we consider all the small types as a single type. Then we have $\text{ALG}(I_i) = \text{OPT}(I_i) \leq \text{OPT}(I)$ for $i = 1, \dots, M-1$, since if items of only one type arrive, our algorithm packs them perfectly. Moreover, $\text{ALG}(I_M) = O(\sqrt{d}) \cdot \text{OPT}(I_M) = O(\sqrt{d}) \cdot \text{OPT}(I)$ for $i = M$. Thus $\text{ALG}(I) = \sum_{i=1}^M \text{ALG}(I_i) \leq (M-1)\text{OPT}(I) + O(\sqrt{d})\text{OPT}(I) = O(d/\log d)\text{OPT}(I)$.

We now prove the lower bound. Consider the following lower bound construction. (This lower bound can also be shown using the weighting function.) We use $\lceil \log d \rceil$ phases. In phase i , $N((2^i - 1)^d - (2^i - 2)^d)$ items of size $2^{-i}(1 + \delta)$ arrive, where $\delta < 2^{-\lceil \log d \rceil} \leq 1/d$. It is possible to place all these items in just N bins by using the following packing scheme. Each bin is packed identically, so we just describe the packing of a single bin. The first item is placed in a corner of the bin. We assign coordinates to the bin so that this corner is the origin and all positive axes are along edges of the bin. (The size of the bin in each dimension is 1.)

Consider any coordinate axis. We reserve the space between $(1 - 2^{1-i})(1 + \delta)$ and $(1 - 2^{-i})(1 + \delta)$ for items of phase i . Note that this is exactly the size of such an item. By doing this along every axis, we can place all $(2^i - 1)^d - (2^i - 2)^d$ items of phase i . (There would be room for $(2^i - 1)^d$ items if we used all the space until $(1 - 2^{-i})(1 + \delta)$ along each axis; we lose $(2^i - 2)^d$ items because the space until $(1 - 2^{1-i})(1 + \delta)$ is occupied.)

The minimum number of bins that any bounded space online algorithm needs to place the items of phase i is $N((2^i - 1)^d - (2^i - 2)^d)/(2^i - 1)^d = N(1 - (\frac{2^i - 2}{2^i - 1})^d)$. Note that the contribution of each phase i to the total number of bins required to pack all items is strictly decreasing in i . Consider the contribution of the last phase, which is phase $\lceil \log d \rceil$. Since $\lceil \log d \rceil \leq 1 + \log d$, it is greater than $N(1 - (\frac{2d-2}{2d-1})^d) = N(1 - (1 - \frac{1}{2d-1})^d) \geq N(1 - e^{-1/2}) > 0.39N$ for all $d \geq 2$. Thus all $\lceil \log d \rceil$ terms all contribute at least $0.39N$, and the total number of bins required is at least $0.39N(\lceil \log d \rceil)$. This implies a lower bound of $\Omega(\log d)$ on the asymptotic performance ratio of this problem. \square

In [14], we give specific upper and lower bounds for dimensions $2, \dots, 7$.

3. Packing hyperboxes. Next, we describe how to extend the algorithm for hypercubes to handle hyperboxes instead of hypercubes. This algorithm also uses the parameter ε . The value of M as a function of ε is picked so that $M \geq 1/(1 - (1 - \varepsilon)^{1/(d+2)}) - 1$. Similar to the previous algorithm, the hyperboxes are classified into types. An arriving hyperbox h of dimensions (h_1, h_2, \dots, h_d) is classified as one of $(2M - 1)^d$ types depending on its components: a type of a hyperbox is the vector of the types of its components.

There are $2M - 1$ types of components. A component larger than $1/M$ has type i if $1/(i + 1) < h_i \leq 1/i$, and is called large. A component smaller than $1/M$ has type i , where $M \leq i \leq 2M - 1$, if there exists a nonnegative integer f_i such that $1/(i + 1) < 2^{f_i} h_i \leq 1/i$. Such components are called small.

Each of the $(2M - 1)^d$ types is packed separately and independently of the others. The algorithm keeps one active bin for each type (s_1, \dots, s_d) . When such a bin is opened, it is split into $\prod_{i=1}^d s_i$ identical sub-bins of dimensions $(1/s_1, \dots, 1/s_d)$. On arrival of a hyperbox h , after classification into a type, a sub-bin has to be found for it. If there is no sub-bin in the current bin that is larger than h in every dimension, we close the bin and open a new one. Otherwise, we take an empty sub-bin that has minimum volume among all sub-bins that can contain h .

Now consider the components of h one by one. If the i th component is large, the sub-bin has the correct size in this dimension: its size is $1/s_i$ whereas the component is in $(1/(s_i + 1), 1/s_i]$.

If the i th component is small, the size of the sub-bin in the i th dimension may be too large. Suppose its size is $1/(2^{f'} s_i)$ whereas the hyperbox has size $\in (1/(2^f (s_i + 1)), 1/(2^f s_i)]$ in this dimension for some $f > f'$. In this case, we divide the sub-bin into two equal parts by cutting halfway (across the i th dimension). If the new sub-bins have the proper size, take one of the two smallest sub-bins that were created, and continue with the next component. Otherwise, take one of the new sub-bins and cut it in half again, repeating until the size of a created sub-bin is $1/(2^f s_i)$.

Thus we ensure that the sub-bin that we use to pack the item h has the proper size in every dimension. We then place this item anywhere inside the sub-bin.

We now generalize the proofs from the previous section for this algorithm.

CLAIM 5. Consider a type (s_1, \dots, s_d) , and its active bin. For every vector $(f_1, \dots, f_d) \neq 0$ of nonnegative integers such that $f_i = 0$ for each large component i , there is at most one empty sub-bin of size $(1/(2^{f_1} s_1), \dots, 1/(2^{f_d} s_d))$.

Proof. Note that the number of sub-bins of size $(1/s_1, \dots, 1/s_d)$, is initialized to be $\prod_{i=1}^d s_i$, and decays until it reaches the value zero. The cutting process does not create more than a single empty sub-bin of each size. This is true for all the sub-bins created except for the smallest size that is created in any given process. For that size we create two identical sub-bins. However, one of them is filled right away.

Furthermore, no sub-bins of existing sizes are created due to the choice of the initial sub-bin. The initial sub-bin is chosen to be of minimum volume among the ones that can contain the item, and hence all the created sub-bins (all of which can contain the item) are of smaller volume than any other existing sub-bin that can contain the item. \square

CLAIM 6. The occupied volume in each closed bin of type (s_1, \dots, s_d) is at least

$$(1 - \varepsilon) \prod_{i \in L} s_i / (s_i + 1),$$

where L is the set of large components in this type.

Proof. To bound the occupied volume in closed bins, note that a sub-bin which was assigned an item is full by a fraction of at least

$$\prod_{i=1}^d \frac{s_i}{s_i + 1} \geq \left(\frac{M}{M + 1} \right)^{d - |L|} \prod_{i \in L} \frac{s_i}{s_i + 1}.$$

Considering sub-bins that were empty when the bin was closed, by Claim 5 there may be one empty sub-bin of each size $(1/(2^{f_1} s_1), \dots, 1/(2^{f_d} s_d))$, with the restrictions

that f_i is a nonnegative integer for $i = 1, \dots, d$, $f_i = 0$ for each large component i , and there exists some $i \in \{1, \dots, d\}$ such that $f_i \neq 0$.

If there are no small components, there can be no empty sub-bins because large components never cause splits into sub-bins, so all sub-bins are used when the bin is closed. This gives a bound of $\prod_{i \in L} s_i / (s_i + 1)$.

If there is only one small component, the total volume of all empty sub-bins that can exist is $1/(s_1 \cdots s_d) \cdot (\frac{1}{2} + \frac{1}{4} + \cdots) \leq 1/(s_1 \cdots s_d) \leq 1/M$, since one of the components is small (type is at least M) and all other components have types of at least 1. The occupied volume is at least $(1 - 1/M) \cdot \frac{M}{M+1} \prod_{i \in L} (s_i / (s_i + 1)) \geq (\frac{M}{M+1})^{d+2} \prod_{i \in L} (s_i / (s_i + 1))$. This holds for any $d \geq 2$ and $M \geq 2$.

If there are $r \geq 2$ small components, the total volume of empty sub-bins is at most $(2^r - 1)/(s_1 s_2 \cdots s_d) \leq (2^r - 1)/M^r \leq 2^r/M^r$. (We get the factor $2^r - 1$ by enumerating over all possible choices of the values f_i .) We get that the fraction of each bin that is filled is at least

$$\begin{aligned} & \left(1 - \frac{2^r}{M^r}\right) \left(\frac{M}{M+1}\right)^r \prod_{i \in L} \frac{s_i}{s_i + 1} \\ &= \frac{M^r - 2^r}{(M+1)^r} \prod_{i \in L} \frac{s_i}{s_i + 1} \geq \left(\frac{M}{M+1}\right)^{r+2} \prod_{i \in L} \frac{s_i}{s_i + 1} \\ &\geq \left(\frac{M}{M+1}\right)^{d+2} \prod_{i \in L} \frac{s_i}{s_i + 1}. \end{aligned}$$

The first inequality holds for $M^r - 2^r \geq M^{r+2}/(M+1)^2$, which holds for any $r \geq 2$ and $M \geq 4$.

Using $(\frac{M}{M+1})^{d+2} \geq 1 - \varepsilon$ we get Claim 6. \square

We now define a weighting function for our algorithm. The weight of a hyperbox p with components (h_1, \dots, h_d) and type (s_1, \dots, s_d) is defined as

$$w_\varepsilon(p) = \frac{1}{1 - \varepsilon} \prod_{i \notin L} h_i \prod_{i \in L} \frac{1}{s_i},$$

where L is the set of large components in this type.

LEMMA 3.1. *For all input sequences σ , $\text{cost}_{\text{alg}}(\sigma) \leq \sum_{h \in \sigma} w_\varepsilon(h) + O(1)$.*

Proof. In order to prove the claim, it is sufficient to show that each closed bin contains items of total weight of at least 1. Consider a bin filled with hyperboxes with type (s_1, \dots, s_d) . It is sufficient to consider the subsequence σ of the input that contains only items of this type, since all types are packed independently. We build an input σ' for which both the behavior of the algorithm and the weights are the same as for σ , and show that the claim holds for σ' . Let $\delta < 1/M^3$ be a very small constant.

For a hyperbox $h \in \sigma$ with components (h_1, \dots, h_d) and type (s_1, \dots, s_d) , let $h' = (h'_1, \dots, h'_d) \in \sigma'$ be defined as follows. For $i \notin L$, $h'_i = h_i$. For $i \in L$, $h'_i = 1/(s_i + 1) + \delta < 1/s_i$. As h and h' have the same type, they require a sub-bin of the same size in all dimensions. Therefore the algorithm packs σ' in the same way as it packs σ . Moreover, according to the definition of weight above, h and h' have the same weight.

Let $v(h)$ denote the volume of an item h . For $h \in \sigma$, we compute the ratio of

weight and volume of the item h' . We have

$$\begin{aligned} \frac{w_\varepsilon(h')}{v(h')} &= \frac{1}{1-\varepsilon} \prod_{i \notin L} h'_i \prod_{i \in L} \frac{1}{s_i} \bigg/ \prod_{i=1}^d h'_i \\ &= \frac{1}{1-\varepsilon} \prod_{i \in L} \frac{1}{s_i h'_i} > \frac{1}{1-\varepsilon} \prod_{i \in L} \frac{s_i + 1}{s_i + M^2 \delta}. \end{aligned}$$

As δ tends to zero, this bound approaches the inverse of the number in Claim 6. This means that the total weight of items in a closed bin is no smaller than 1. \square

Just like in section 2, Lemma 3.1 implies that the asymptotic worst case ratio is upper bounded by the maximum amount of weight that can be packed in a single bin. After the following definition, we prove a technical lemma which implies that this weighting function is also “optimal” in that it determines the true asymptotic performance ratio of our algorithm.

DEFINITION 3.2. *The pseudovolume of a hyperbox $h = (h_1, \dots, h_d)$ is defined as $\prod_{i \notin L} h_i$, where L is the set of large components of h .*

Suppose that for a given set of hyperboxes X , we can partition the dimensions into two sets, S and T , such that for each dimension j in S , we have that the j th components of all hyperboxes in X are bounded in an interval $(1/(k_j + 1), 1/k_j]$. There are no restrictions on the dimensions in T . (Thus such a partition can always be found by taking $S = \emptyset$.)

For a hyperbox $h \in X$, define the *generalized pseudovolume* of the components in T by $\tilde{v}(h, T) = \prod_{j \in T} h_j$, where h_j is the j th component of h . Define the total generalized pseudovolume of all hyperboxes in a set X by $\tilde{v}(X, T) = \sum_{h \in X} \tilde{v}(h, T)$.

CLAIM 7. *For a given set X of hyperboxes, for sufficiently large N , any packing of X into bins requires at least $\tilde{v}(X, T)(1 - \frac{1}{N})^{|T|} / \prod_{i \in S} k_i$ bins, where S and T form a partitioning of the dimensions as described above.*

Proof. We prove the claim by induction on the number of dimensions in T . For $|T| = 0$, we find that the total generalized pseudovolume of X is simply the number of hyperboxes in X (since the empty product is 1) and thus the claim is true using Claim 4.

Assume the claim is true for $|T| = 0, \dots, r - 1$. Suppose $|S| = d - r < d$. Take any dimension $i \in T$. We replace each hyperbox h , with component h_i in dimension i , by $\lfloor N^2 h_i \rfloor$ hyperboxes that have $\frac{1}{N^2}$ as their i th component, and are identical to h in all other components. Here N is taken sufficiently large, such that $\frac{1}{N} < h_i$. Clearly, the new input X' is no harder to pack, as we split each item into parts whose sum is smaller than or equal to the original items. The total generalized pseudovolume of the hypercubes in X' is at most a factor of $1 - \frac{1}{N^2 h_i} \geq 1 - \frac{1}{N}$ smaller than that of X . So if we write $T' = T \setminus \{i\}$, we have $\tilde{v}(X', T') \cdot \frac{1}{N^2} \geq \tilde{v}(X, T)(1 - \frac{1}{N})$. By induction, it takes at least

$$\tilde{v}(X', T') \cdot \left(1 - \frac{1}{N}\right)^{r-1} / \prod_{j \in S \cup \{i\}} k_j$$

bins to pack the modified input X' . Using that $k_i = N^2$, this is $\tilde{v}(X, T) \cdot (1 - \frac{1}{N})^r / \prod_{j \in S} k_j$ bins. \square

Letting $\gamma = 1 - (1 - \frac{1}{N})^d$, we get that the required number is at least $\tilde{v}(X, T)(1 - \gamma) / \prod_{j \in S} k_j$ bins, where $\gamma \rightarrow 0$ as $N \rightarrow \infty$. In the remainder, we will take S to be the dimensions where the components of the hyperboxes in X are large, and T the

dimensions where they are small. Note that this choice of S satisfies the constraints on S above, and that this reduces the generalized pseudovolume to the (normal) pseudovolume defined before. We are ready to prove the following lemma.

LEMMA 3.3. *Let $\varepsilon > 0$. Suppose the maximum amount of weight that can be packed in a single bin is α_ε . Then our algorithm has an asymptotic performance ratio of α_ε , and the asymptotic performance ratio of any bounded space algorithm is at least $(1 - \varepsilon)\alpha_\varepsilon$.*

Proof. The first statement follows from Lemma 3.1. We show a lower bound of value which tends to α_ε on the asymptotic performance ratio of any bounded space algorithm.

Consider a packed bin for which the sum of weights is α_ε . Partition the hyperboxes of this bin into M^d types in the following way. Each component is either of a type in $\{1, \dots, M - 1\}$ or small (i.e., of a type i , $i \leq M$). Let N' be a large constant. The sequence consists of phases. Each phase consists of one item from the packed bin, repeated N' times. The optimal offline cost is therefore N' . Using Claim 7 we see that the amount of bins needed to pack a phase which consists of an item p repeated N' times is simply $N'w_\varepsilon(p)(1 - \gamma)(1 - \varepsilon)$. Therefore the cost of an online algorithm is at least $N'\alpha_\varepsilon(1 - \gamma)(1 - \varepsilon) - O(1)$, which makes the asymptotic performance ratio arbitrarily close to $(1 - \varepsilon)\alpha_\varepsilon$. \square

Furthermore, we can determine the asymptotic performance ratio of our algorithm for hyperbox packing. Comparing to the unbounded space algorithm in [11] we can see that all the weights we defined are smaller than or equal to the weights used in [11]. So the asymptotic performance ratio is not higher. However, it can also not be lower due to the general lower bound for bounded space algorithms. This means that both algorithms have the same asymptotic performance ratio, namely $(\Pi_\infty)^d$, where $\Pi_\infty \approx 1.691$ is the asymptotic performance ratio of the algorithm HARMONIC [25].

4. Variable-sized packing. In this section we consider the problem of multi-dimensional packing where the bins used can have different sizes. We assume that all bins are hypercubes, with sides $\alpha_1 < \alpha_2 < \dots < \alpha_m = 1$. In fact our algorithm is more general and works for the case where the bins are hyperboxes with dimensions α_{ij} ($i = 1, \dots, m$, $j = 1, \dots, d$). We present the special case of bins that are hypercubes in this paper in order to avoid an overburdened notation and messy technical details.

The main structure of the algorithm is identical to the one in section 3. The main problem in adapting that algorithm to the current problem is selecting the right bin size to pack the items in. In the one-dimensional variable-sized bin packing problem, it is easy to see which bin will accommodate any given item the best; here it is not so obvious how to select the right bin size, since in one dimension a bin of a certain size might seem best whereas for other dimensions, other bins seem more appropriate.

We begin by defining types for hyperboxes based on their components and the available bin sizes. Once again we use a parameter ε . The value of M as a function of ε is again picked so that $M \geq 1/(1 - (1 - \varepsilon)^{1/(d+2)}) - 1$. An arriving hyperbox h of dimensions (h_1, h_2, \dots, h_d) is classified as one of at most $(2mM/\alpha_1 - 1)^d$ types depending on its components: a type of a hyperbox is the vector of the types of its components. We define

$$T_i = \left\{ \frac{\alpha_i}{j} \mid j \in \mathbb{N}, \frac{\alpha_i}{j} \geq \frac{\alpha_1}{2M} \right\}, \quad T = \bigcup_{i=1}^m T_i.$$

Let the members of T be $1 = t_1 > t_2 > \dots > t_{q'} = \alpha_1/M > \dots > t_q = \alpha_1/(2M)$.

The interval I_j is defined to be $(t_{j+1}, t_j]$ for $j = 1, \dots, q'$. Note that these intervals are disjoint and they cover $(\alpha_1/M, 1]$.

A component larger than α_1/M has type i if $h_i \in I_i$, and is called large. A component smaller than α_1/M has type i , where $q' \leq i \leq q - 1$, if there exists a non-negative integer f_i such that $t_{i+1} < 2^{f_i} h_i \leq t_i$. Such components are called small. Thus in total there are $q - 1 \leq 2mM/\alpha_1 - 1$ component types.

Bin selection. We now describe how to select a bin for a given type. Intuitively, the size of this bin is chosen in order to maximize the number of items packed relative to the area used. This is done as follows.

For a given component type s_i and bin size α_j , write $F(s_i, \alpha_j) = \max\{k \mid \alpha_j/k \geq t_{s_i}\}$. Thus for a large component, $F(s_i, \alpha_j)$ is the number of times that a component of type s_i fits in an interval of length α_j . This number is uniquely defined due to the definition of the numbers t_i . Basically, the general classification into types is too fine for any particular bin size, and we use $F(s_i, \alpha_j)$ to get a less refined classification which only considers the points t_i of the form α_j/k .

Denote by L the set of components in type $s = (s_1, \dots, s_d)$ that are large. If $L = \emptyset$, we use a bin of size 1 for this type. Otherwise, we place this type in a bin of any size α_j which maximizes¹

$$(2) \quad \prod_{i \in L} \frac{F(s_i, \alpha_j)}{\alpha_j}.$$

Thus we do not take small components into account in this formula. Note that for a small component, $F(s_i, \alpha_j)$ is not necessarily the same as the number of times that such a component fits into any interval of length α_j . However, it is at least M for any small component.

When such a bin is opened, it is split into $\prod_{i=1}^d F(s_i, \alpha_j)$ identical sub-bins of dimensions $(\alpha_j/F(s_1, \alpha_j), \dots, \alpha_j/F(s_d, \alpha_j))$. These bins are then further sub-divided into sub-bins in order to place hyperboxes in “well-fitting” sub-bins in the manner which is described in section 3.

Similar to in section 3, the following claim can now be shown.

CLAIM 8. *The occupied volume in each closed bin of type $s = (s_1, \dots, s_d)$ is at least*

$$V_{s,j} = (1 - \varepsilon) \alpha_j^d \prod_{i \in L} \frac{F(s_i, \alpha_j)}{F(s_i, \alpha_j) + 1},$$

where L is the set of large components in this type and α_j is the bin size used to pack this type.

We now define a weighting function for our algorithm. The weight of a hyperbox h with components (h_1, \dots, h_d) and type $s = (s_1, \dots, s_d)$ is defined as

$$w_\varepsilon(h) = \frac{1}{1 - \varepsilon} \left(\prod_{i \notin L} h_i \right) \left(\prod_{i \in L} \frac{\alpha_j}{F(s_i, \alpha_j)} \right),$$

where L is the set of large components in this type and α_j is the size of bins used to pack this type.

¹For the case that the bins are hyperboxes instead of hypercubes, we get the formula $\prod_{i \in L} (F(s_i, \alpha_{ij})/\alpha_{ij})$, and similar changes throughout the text.

In order to prove that this weighting function works (gives a valid upper bound for the cost of our algorithm), we will want to modify components s_i to the smallest possible component such that $F(s_i, \alpha_j)$ does not change. (Basically, a component will be rounded to $\alpha_j/(F(s_i, \alpha_j) + 1)$ plus a small constant.) However, with variable-sized bins, when we modify components in this way, the algorithm might decide to pack the new hyperbox differently. (Remember that $F(s_i, \alpha_j)$ is a “less refined” classification which does not take other bin sizes than α_j into account.) To circumvent this technical difficulty, we will show first that as long as the algorithm keeps using the same bin size for a given item, the volume guarantee still holds.

For a given type $s = (s_1, \dots, s_d)$ and the corresponding set L and bin size α_j , define an *extended type* $\text{Ext}(s_1, \dots, s_d)$ as follows: an item h is of extended type $\text{Ext}(s_1, \dots, s_d)$ if each large component $h_i \in (\frac{\alpha_j}{F(s_i, \alpha_j)+1}, \frac{\alpha_j}{F(s_i, \alpha_j)}]$ and each small component h_i is of type s_i .

COROLLARY 4.1. *Suppose items of extended type $\text{Ext}(s_1, \dots, s_d)$ are packed into bins of size α_j . Then the occupied volume in each closed bin is at least $V_{s,j}$.*

Proof. In the proof of Claim 8, we only use that each large component h_i is contained in the interval $(\frac{\alpha_j}{F(s_i, \alpha_j)+1}, \frac{\alpha_j}{F(s_i, \alpha_j)}]$. Thus the proof also works for extended types. \square

LEMMA 4.2. *For all input sequences σ , $\text{cost}_{\text{alg}}(\sigma) \leq \sum_{h \in \sigma} w_\varepsilon(h) + O(1)$.*

Proof. In order to prove the claim, it is sufficient to show that each closed bin of size α_j contains items of total weight of at least α_j^d . Consider a bin of this size filled with hyperboxes of type $s = (s_1, \dots, s_d)$. It is sufficient to consider the subsequence σ of the input that contains only items of this type, since all types are packed independently. This subsequence only uses bins of size α_j so we may assume that *no other sizes of bins are given*. We build an input σ' for which both the behavior of the algorithm and the weights are the same as for σ and show the claim holds for σ' . Let $\delta < 1/M^3$ be a very small constant.

For a hyperbox $h \in \sigma$ with components (h_1, \dots, h_d) and type $s = (s_1, \dots, s_d)$, let $h' = (h'_1, \dots, h'_d) \in \sigma'$ be defined as follows. For $i \notin L$, $h'_i = h_i$. For $i \in L$, $h'_i = \alpha_j/(F(s_i, \alpha_j) + 1) + \delta < \alpha_j/F(s_i, \alpha_j)$.

Note that h' is of extended type $\text{Ext}(s_1, \dots, s_d)$. Since only one bin size is given, the algorithm packs σ' in the same way as it packs σ . Moreover, according to the definition of weight above, h and h' have the same weight.

Let $v(h)$ denote the volume of an item h . For $h \in \sigma$, we compute the ratio of weight and volume of the item h' . We have

$$\begin{aligned} \frac{w_\varepsilon(h')}{v(h')} &= \frac{w_\varepsilon(h)}{v(h')} = \frac{1}{1-\varepsilon} \left(\prod_{i \notin L} h'_i \right) \left(\prod_{i \in L} \frac{\alpha_j}{F(s_i, \alpha_j)} \right) \Big/ \prod_{i=1}^d h'_i \\ &= \frac{1}{1-\varepsilon} \prod_{i \in L} \frac{\alpha_j}{F(s_i, \alpha_j)h'_i} > \frac{1}{1-\varepsilon} \prod_{i \in L} \frac{F(s_i, \alpha_j) + 1}{F(s_i, \alpha_j) + M \frac{\alpha_j}{\alpha_1} \delta}. \end{aligned}$$

Here we have used in the last step that a component with a large type fits less than M times in a (one-dimensional) bin of size α_1 , and therefore less than $M \frac{\alpha_j}{\alpha_1}$ times in a bin of size $\alpha_j \geq \alpha_1$. As δ tends to zero, this bound approaches $\alpha_j^d/V_{s,j}$. We find

$$w_\varepsilon(h) \geq \alpha_j^d \frac{v(h')}{V_{s,j}} \quad \text{for all } h \in \sigma.$$

Then Corollary 4.1 implies that the total weight of items in a closed bin of size α_j is no smaller than α_j^d , which is the cost of such a bin. \square

Suppose the optimal solution for a given input sequence σ uses n_j bins of size α_j . Denote the i th bin of size α_j by $B_{i,j}$. Then

$$\frac{\sum_{h \in \sigma} w_\varepsilon(h)}{\sum_{j=1}^m \alpha_j^d n_j} = \frac{\sum_{j=1}^m \sum_{i=1}^{n_j} \sum_{h \in B_{i,j}} w_\varepsilon(h)}{\sum_{j=1}^m \alpha_j^d n_j} = \frac{\sum_{j=1}^m \sum_{i=1}^{n_j} \sum_{h \in B_{i,j}} w_\varepsilon(h)}{\sum_{j=1}^m \sum_{i=1}^{n_j} \alpha_j^d}.$$

This implies that the asymptotic worst case ratio is upper bounded by

$$(3) \quad \max_j \max_{X_j} \sum_{h \in X_j} w_\varepsilon(h) / \alpha_j^d,$$

where the second maximum is taken over all sets X_j that can be packed in a bin of size α_j . Similar to section 3, it can now be shown that this weighting function is also “optimal” in that it determines the true asymptotic performance ratio of our algorithm.

In particular, it can be shown that packing a set of hyperboxes X that have the same type vectors of large and small dimensions takes at least

$$\sum_{h \in X} \prod_{i \notin L} \frac{h_i}{\alpha_j} \bigg/ \prod_{i \in L} F(s_i, \alpha_j)$$

bins of size α_j , where h_i is the i th component of hyperbox h , s_i is the type of the i th component, and L is the set of large components (for all the hyperboxes in X). Since the cost of such a bin is α_j^d , this means that the total cost to pack N' copies of some item h is at least $N'w_\varepsilon(h)(1 - \varepsilon)$ when bins of this size are used. However, it is clear that using bins of another size α_k does not help: packing N' copies of h into such bins would give a total cost of

$$N' \left(\prod_{i \notin L} h_i \right) \left(\prod_{i \in L} \frac{\alpha_k}{F(s_i, \alpha_k)} \right).$$

Since α_j was chosen to maximize $\prod_{i \in L} (F(s_i, \alpha_j) / \alpha_j)$, this expression cannot be less than $N'w_\varepsilon(h)(1 - \varepsilon)$. More precisely, any bins that are not of size α_j can be replaced by the appropriate number of bins of size α_j without increasing the total cost by more than 1 (it can increase by 1 due to rounding).

This implies that our algorithm is optimal among online bounded space algorithms.

5. Resource augmented packing. The resource augmented problem is now relatively simple to solve. In this case, the online algorithm has bins at its disposal that are hypercubes of dimensions $b_1 \times b_2 \times \dots \times b_d$. We can use the algorithm from section 3 with the following modification: the types for dimension j are not based on intervals of the form $(1/(i + 1), 1/i]$ but rather intervals of the form $(b_j/(i + 1), b_j/i]$.

Then, to pack items of type $s = (s_1, \dots, s_d)$, a bin is split into $\prod_{i=1}^d s_i$ identical sub-bins of dimensions $(b_1/s_1, \dots, b_d/s_d)$, and then subdivided further as necessary.

We now find that each closed bin of type $s = (s_1, \dots, s_d)$ is full by at least

$$(1 - \varepsilon)B \prod_{i \in L} \frac{s_i}{s_i + 1},$$

where L is the set of large components in this type, and $B = \prod_{j=1}^d b_j$ is the volume of the bins of the online algorithm.

We now define the weight of a hyperbox h with components (h_1, \dots, h_d) and type $s = (s_1, \dots, s_d)$ as

$$w_\varepsilon(h) = \frac{1}{1 - \varepsilon} \left(\prod_{i \notin L} \frac{h_i}{b_i} \right) \left(\prod_{i \in L} \frac{1}{s_i} \right),$$

where L is the set of large components in this type.

This can be shown to be valid similarly as before, and it can also be shown that items cannot be packed better. However, in this case we are additionally able to give explicit bounds for the asymptotic performance ratio.

5.1. The asymptotic performance ratio. Csirik and Woeginger [13] showed the following for the one-dimensional case.

For a given bin size b , define an infinite sequence $T(b) = \{t_1, t_2, \dots\}$ of positive integers as follows:

$$t_1 = \lfloor 1 + b \rfloor \quad \text{and} \quad r_1 = \frac{1}{b} - \frac{1}{t_1},$$

and for $i = 1, 2, \dots$, let

$$t_{i+1} = \left\lfloor 1 + \frac{1}{r_i} \right\rfloor \quad \text{and} \quad r_{i+1} = r_i - \frac{1}{t_{i+1}}.$$

Define

$$\rho(b) = \sum_{i=1}^{\infty} \frac{1}{t_i - 1}.$$

LEMMA 5.1. *For every bin size $b \geq 1$, there exist online bounded space bin packing algorithms with worst case performance arbitrarily close to $\rho(b)$. For every bin size $b \geq 1$, the bound $\rho(b)$ cannot be beaten by any online bounded space bin packing algorithm.*

The following lemma was proved in Csirik and Van Vliet [11] for a specific weighting function which is independent of the dimension, and is similar to a result of Li and Cheng [26]. However, the proof holds for any positive one-dimensional weighting function w . We extend it for the case where the weighting function depends on the dimension. For a one-dimensional weighting function w_j and an input sequence σ , define $w_j(\sigma) = \sum_{h \in \sigma} w_j(h)$. Furthermore, define $W_j = \sup_{\sigma} w_j(\sigma)$, where the supremum is taken over all sequences that can be packed into a one-dimensional bin.

LEMMA 5.2. *Let σ be a list of d -dimensional rectangles, and let Q be a packing which packs these rectangles into a d -dimensional unit cube. Let w_j ($j = 1, \dots, d$) be arbitrary one-dimensional weighting functions. For each $h \in \sigma$, we define a new hyperbox h' as follows: $s_j(h') = w_j(s_j(h))$ for $1 \leq j \leq d$. Denote the resulting list of hyperboxes by σ' . Then, there exists a packing Q' which packs σ' into a cube of size (W_1, \dots, W_d) .*

Proof. We use a construction analogous to the one in [11]. We transform the packing $Q = Q^0$ of σ into a packing Q^d of σ' in a cube of the desired dimensions. This is done in d steps, one for each dimension. Denote the coordinates of item h in packing Q^i by $(x_1^i(h), \dots, x_d^i(h))$, and its dimensions by $(s_1^i(h), \dots, s_d^i(h))$.

In step i , the coordinates as well as the sizes in dimension i are adjusted as follows. First, we adjust the sizes and set $s_i^i(h) = w_i(s_i(h))$ for every item h , leaving other dimensions unchanged.

To adjust coordinates, for each item h in packing Q^{i-1} we find the “left-touching” items, which is the set of items g which overlap with h in $d-1$ dimensions, and for which $x_i^{i-1}(g) + s_i^{i-1}(g) = x_i^{i-1}(h)$. We may assume that for each item h , there is either a left-touching item or $x_i^{i-1}(h) = 0$.

Then, for each item h that has no left-touching items, we set $x_i^i(h) = 0$. For all other items h , starting with the ones with smallest i -coordinate, we make the i -coordinate equal to $\max(x_i^i(g) + s_i^i(g))$, where the maximum is taken over the left-touching items of h in packing S^{i-1} . Note that we use the new coordinates and sizes of left-touching items in this construction and that this creates a packing without overlap.

If in any step i the items need more than W_i room, this implies a chain of left-touching items with total size less than 1 but total weight more than W_i . From this we can find a set of one-dimensional items that fit in a bin but have total weight more than W_i (using weighting function w_i), which is a contradiction. \square

As in [11], this implies immediately that the total weight that can be packed into a unit-sized bin is upper bounded by $\prod_{i=1}^d W_i$, which in the present case is $\prod_{i=1}^d \rho(b_i)$. Moreover, by extending the lower bound from [13] to d dimensions exactly as in [11], it can be seen that the asymptotic performance ratio of any online bounded space bin packing algorithm can also not be lower than $\prod_{i=1}^d \rho(b_i)$.

6. Conclusions. An open question left by this paper is what the asymptotic performance ratio of the bounded space hypercube packing problem is. We can show that it is $\Omega(\log d)$, and we conjecture that it is $\Theta(\log d)$. Giving an explicit expression for the competitive ratio in variable-sized packing (as a function of the bin sizes) would be harder. Already in [30] where an optimal one-dimensional bounded space algorithm was given for the variable-sized problem, its ratio is unknown. It is interesting to find out whether in the multidimensional case the worst case occurs when only unit sized bins are available.

REFERENCES

- [1] N. BANSAL AND M. SVIRIDENKO, *New approximability and inapproximability results for 2-dimensional packing*, in Proceedings of the 15th Annual Symposium on Discrete Algorithms, ACM, New York, SIAM, Philadelphia, 2004, pp. 189–196.
- [2] D. BLITZ, A. VAN VLIET, AND G. J. WOEGERING, *Lower Bounds on the Asymptotic Worst-case Ratio of Online Bin Packing Algorithms*, Unpublished manuscript, 1996.
- [3] D. J. BROWN, *A Lower Bound for On-line One-dimensional Bin Packing Algorithms*, Technical report R-864, Coordinated Sci. Lab., Urbana, IL, 1979.
- [4] A. CAPRARA, *Packing 2-dimensional bins in harmony*, in Proceedings of the 43th Annual IEEE Symposium on Foundations of Computer Science, 2002, pp. 490–499.
- [5] F. R. K. CHUNG, M. R. GAREY, AND D. S. JOHNSON, *On packing two-dimensional bins*, SIAM J. Algebraic Discrete Methods, 3 (1982), pp. 66–76.
- [6] E. G. COFFMAN, M. R. GAREY, AND D. S. JOHNSON, *Approximation algorithms for bin packing: A survey*, Approximation Algorithms, D. Hochbaum, ed., PWS Publishing Company, Boston, 1997.
- [7] D. COPPERSMITH AND P. RAGHAVAN, *Multidimensional on-line bin packing: Algorithms and worst case analysis*, Oper. Res. Lett., 8 (1989), pp. 17–20.
- [8] J. CORREA AND C. KENYON, *Approximation schemes for multidimensional packing*, in Proceedings of the 15th Annual ACM/SIAM Symposium on Discrete Algorithms, ACM, New York, SIAM, Philadelphia, 2004, pp. 179–188.
- [9] J. CSIRIK, *An on-line algorithm for variable-sized bin packing*, Acta Inf., 26 (1989), pp. 697–709.
- [10] J. CSIRIK, J. B. G. FRENK, AND M. LABBE, *Two-dimensional rectangle packing: On-line methods and results*, Discrete Appl. Math., 45 (1993), pp. 197–204.
- [11] J. CSIRIK AND A. VAN VLIET, *An on-line algorithm for multidimensional bin packing*, Oper. Res. Lett., 13 (1993), pp. 149–158.

- [12] J. CSIRIK AND G. J. WOEGINGER, *On-line packing and covering problems*, Online Algorithms: The State of the Art, Lecture Notes in Comput. Sci. 1442, A. Fiat and G. J. Woeginger, eds., Springer-Verlag, Berlin, 1998, pp. 147–177.
- [13] J. CSIRIK AND G. J. WOEGINGER, *Resource augmentation for online bounded space bin packing*, in Proceedings of the 27th Annual International Colloquium on Automata, Languages, and Programming, Geneva, Switzerland, 2000, pp. 296–304.
- [14] L. EPSTEIN AND R. VAN STEE, *Bounds for Online Bounded Space Hypercube Packing*, Technical report SEN-E0417, CWI, Amsterdam, The Netherlands, 2004.
- [15] W. FERNANDEZ DE LA VEGA AND G. S. LUEKER, *Bin packing can be solved within $1 + \epsilon$ in linear time*, *Combinatorica*, 1 (1981), pp. 349–355.
- [16] C. E. FERREIRA, F. K. MIYAZAWA, AND Y. WAKABAYASHI, *Packing squares into squares*, *Pesquisa Operacional*, 19 (1999), pp. 223–237.
- [17] D. K. FRIESEN AND M. A. LANGSTON, *Variable sized bin packing*, *SIAM J. Comput.*, 15 (1986), pp. 222–230.
- [18] G. GALAMBOS, *A 1.6 lower bound for the two-dimensional on-line rectangle bin packing*, *Acta Cybernet.*, 10 (1991), pp. 21–24.
- [19] G. GALAMBOS AND A. VAN VLIET, *Lower bounds for 1-, 2-, and 3-dimensional on-line bin packing algorithms*, *Computing*, 52 (1994), pp. 281–297.
- [20] M. R. GAREY, R. L. GRAHAM, AND J. D. ULLMAN, *Worst-case analysis of memory allocation algorithms*, in Proceedings of the Fourth Annual ACM Symposium on Theory of Computing, ACM, Denver, CO, 1972, pp. 143–150.
- [21] D. S. JOHNSON, *Near-optimal Bin Packing Algorithms*, Ph.D. thesis, MIT, Cambridge, MA, 1973.
- [22] D. S. JOHNSON, *Fast algorithms for bin packing*, *J. Comput. System Sci.*, 8 (1974), pp. 272–314.
- [23] N. KARMARKAR AND R. M. KARP, *An efficient approximation scheme for the one-dimensional bin-packing problem*, in Proceedings of the 23rd Annual Symposium on Foundations of Computer Science, Chicago, 1982, pp. 312–320.
- [24] Y. KOHAYAKAWA, F. K. MIYAZAWA, P. RAGHAVAN, AND Y. WAKABAYASHI, *Multidimensional cube packing*, *Electronic Notes in Discrete Mathematics*, Vol. 7, Jayme Szwarcfiter and Siang Song, eds., Elsevier Science Publishers, Amsterdam, The Netherlands, 2001.
- [25] C. C. LEE AND D. T. LEE, *A simple on-line bin packing algorithm*, *J. ACM*, 32 (1985), pp. 562–572.
- [26] K. LI AND K. H. CHENG, *Generalized first-fit algorithms in two and three dimensions*, *Internat. J. Found. Comput. Sci.*, 1 (1990), pp. 131–150.
- [27] K. LI AND K.-H. CHENG, *A Generalized Harmonic Algorithm for On-line Multi-dimensional Bin Packing*, Technical report UH-CS-90-2, University of Houston, Houston, TX, 1990.
- [28] F. M. LIANG, *A lower bound for on-line bin packing*, *Inform. Process. Lett.*, 10 (1980), pp. 76–79.
- [29] F. K. MIYAZAWA AND Y. WAKABAYASHI, *Cube packing*, *Theoret. Comput. Sci.*, 297 (2003), pp. 355–366.
- [30] S. S. SEIDEN, *An optimal online algorithm for bounded space variable-sized bin packing*, *SIAM J. Discrete Math.*, 14 (2001), pp. 458–470.
- [31] S. S. SEIDEN, *On the online bin packing problem*, *J. ACM*, 49 (2002), pp. 640–671.
- [32] S. S. SEIDEN AND R. VAN STEE, *New bounds for multidimensional packing*, *Algorithmica*, 36 (2003), pp. 261–293.
- [33] J. D. ULLMAN, *The Performance of a Memory Allocation Algorithm*, Technical report 100, Princeton University, Princeton, NJ, 1971.
- [34] A. VAN VLIET, *An improved lower bound for on-line bin packing algorithms*, *Inform. Process. Lett.*, 43 (1992), pp. 277–284.
- [35] A. VAN VLIET, *Lower and Upper Bounds for Online Bin Packing and Acheduling Heuristics*, Ph.D. thesis, Erasmus University, Rotterdam, The Netherlands, 1995.
- [36] A. VAN VLIET, *On the asymptotic worst case behavior of harmonic fit*, *J. Algorithms*, 20 (1996), pp. 113–136.
- [37] G. J. WOEGINGER, *Improved space for bounded-space online bin packing*, *SIAM J. Discrete Math.*, 6 (1993), pp. 575–581.
- [38] A. C. C. YAO, *New algorithms for bin packing*, *J. ACM*, 27 (1980), pp. 207–227.

STRICTLY NONBLOCKING WDM CROSS-CONNECTS*

APRIL RASALA[†] AND GORDON WILFONG[‡]

Abstract. Using wavelength division multiplexing (WDM) technology, an optical network can route multiple signals simultaneously along a single optical fiber by encoding each signal on its own wavelength. If the network contains places where multiple fibers connect together and signals are allowed to be moved from any of the incoming fibers to any of the outgoing fibers, then the network is said to contain *cross-connects*. More precisely, a $k_1 \times k_2$ WDM *cross-connect* has k_1 input fibers and k_2 output fibers. Each of the k_1 input fibers supports the same n_1 input wavelengths and each of the k_2 output fibers supports the same n_2 output wavelengths. Since a signal on input wavelength λ can be routed from its input fiber to an output fiber such that it arrives on the output fiber using wavelength γ , where $\lambda \neq \gamma$, the cross-connect must be capable of performing wavelength conversion. Along any fiber in the cross-connect a device called a *wavelength interchanger* can be inserted to perform wavelength conversion. In other words if the path of a signal from an input fiber to an output fiber passes through a wavelength interchanger, then the wavelength of the signal can be changed to any wavelength that is not already in use along the fiber leaving the wavelength interchanger. Given the high cost of wavelength interchangers, the overall cost of a $k_1 \times k_2$ WDM cross-connect is minimized by reducing the number of wavelength interchangers in the cross-connect. However, a desirable property for a cross-connect C is for C to always be able to provide a route (and wavelength conversion) for any valid demand from any pair of input and output fibers regardless of the routes of other demands currently routed in C . If C has this capability then it is said to be *strictly nonblocking*.

For most of this paper we consider a demand to be a request for a connection from an input fiber to an output fiber such that the connection starts on a specified input wavelength and leaves the cross-connect on a second specified wavelength. Using this demand model, we consider cross-connects for which k_1 is not necessarily equal to k_2 and the number n_1 of supported input wavelengths can differ from the number n_2 of supported output wavelengths. Without loss of generality we assume that $k_1 \leq k_2$ and present a family of strictly nonblocking $k_1 \times k_2$ WDM cross-connects that use $\min(k_1 + k_2 - 1, n_1 k_1)$ wavelength interchangers. For the case when $k_1 = k_2 = k$ and $n_1 = n_2$, we prove that this is optimal. For cross-connects where n_1 is not necessarily equal to n_2 , we show that if there is at most one wavelength interchanger on any path from an input fiber to an output fiber, $\min(k_1 + k_2 - 1, n_1 k_1)$ wavelength interchangers are optimal. Finally, we consider a more flexible demand model where $k_1 = k_2$ but the input and output wavelengths are not specified as part of the demand. We show that $2k - 1$ wavelength interchangers are still necessary for any strictly nonblocking $k \times k$ WDM cross-connect.

Key words. optical networking, cross-connect, wavelength division multiplexing, strictly non-blocking

AMS subject classifications. 90B18, 68M10

DOI. 10.1137/S0097539703434255

1. Introduction. We present designs of cross-connects for optical networks supporting wavelength division multiplexing. We prove that our designs use only the minimum number of an expensive device, known as a wavelength interchanger, and thus are optimal in terms of the cost of the cross-connect.

A fiber in an optical network supporting wavelength division multiplexing (WDM) can carry multiple signals simultaneously if each signal is encoded on a distinct wave-

*Received by the editors September 9, 2003; accepted for publication (in revised form) April 19, 2005; published electronically November 9, 2005. Early versions of some of this work appeared in STOC 2000 and SODA 2000.

<http://www.siam.org/journals/sicomp/35-2/43425.html>

[†]MIT Computer Science and Artificial Intelligence Laboratory, Cambridge, MA 02139 (arasala@theory.csail.mit.edu). The work of this author was supported by a Lucent GRPW Fellowship.

[‡]Bell Labs, Lucent Technologies, Murray Hill, NJ 07974 (gtw@research.bell-labs.com).

length. In order to fully exploit the benefits of WDM technology, it is useful to be able to switch a signal from one wavelength to another when the signal switches to a new fiber. A $k_1 \times k_2$ WDM cross-connect has k_1 input fibers and k_2 output fibers and allows signals from the input fibers to be moved to available wavelengths on the output fibers. Specifically, a *demand* on the cross-connect specifies an input fiber, an input wavelength, an output fiber and an output wavelength. If the input wavelength is unused on the input fiber and the output wavelength is unused on the output fiber, then the cross-connect should be able to route the signal from the input fiber to the output fiber *and* change the encoding of the signal from the input wavelength to the output wavelength. A WDM cross-connect must maintain current connections (the results of previous demands placed on it) and at the same time satisfy new requests for connections between input fibers and output fibers. A *strictly nonblocking* cross-connect C is guaranteed to be able to route any new demand placed on it regardless of the set of existing connections routed through C .

A cross-connect contains various optical devices and optical fibers. A wavelength interchanger is an optical device with a single input fiber and a single output fiber. Any of the signals entering the wavelength interchanger on any of the supported wavelengths can be changed to any of the supported wavelengths on the fiber leaving the wavelength interchanger provided those wavelengths are not already in use by another connection. Given the high cost of wavelength interchangers now and in the foreseeable future [13], the cost of a strictly nonblocking WDM cross-connect is dominated by the number of wavelength interchangers required in the design. Thus we focus our attention on determining the optimal number of wavelength interchangers needed in any strictly nonblocking $k_1 \times k_2$ WDM cross-connect and on presenting a family of strictly nonblocking $k_1 \times k_2$ WDM cross-connects that use the optimal number of wavelength interchangers.

The task of optimizing strictly nonblocking WDM cross-connects is simplified by drawing from work on classical cross-connects. In classical cross-connects, signals do not share wires and therefore there is no need for wavelength conversion. Of course, one can view time slots in a time division multiplexing (TDM) network as equivalent to a WDM network but time slot interchange is an inexpensive operation as opposed to wavelength interchange in a WDM network. A vast amount of literature exists on the design of such cross-connects [4, 8, 12, 1, 7]. In particular, work has been done on the optimization of strictly nonblocking classical cross-connects that guarantee that a connection can always be made between any input line and any output line provided neither are already in use. The topology of a strictly nonblocking classical cross-connect can be used as the topology of a WDM network. In this case we assume that the WDM network supports n wavelength and does not allow any wavelength conversion. Since the WDM network has the topology of a classical network and does not support wavelength conversion, it is conceptually equivalent to n copies, one for each wavelength, of the classical network. Furthermore, since the WDM network has the topology of a strictly nonblocking classical cross-connect, any connection on any wavelength can be completed as long as the requesting input and output fibers do not already have an existing connection on that wavelength. We refer to a section of directed fiber and optical devices without wavelength conversion capability as a *fabric*.

The designs of strictly nonblocking WDM cross-connects presented in this paper use the topology of classical strictly nonblocking cross-connects as the basis for fabrics that connect the input fibers to a set of wavelength interchangers and connect those wavelength interchangers to the set of output fibers. Such a design is referred to

as a split cross-connect. For the case where the number of input fibers, k , is equal to the number of output fibers and the set of input wavelengths is the same as the set of output wavelengths, we present split cross-connects with $2k - 1$ wavelength interchangers and prove that these designs are optimal. We also consider general split cross-connects where the number of input fibers, k_1 , and the number of input wavelengths, n_1 , may differ from the number of output fibers, k_2 , and the number of output wavelengths, n_2 . In this case we present designs of strictly nonblocking split cross-connects that use $\min(k_1 + k_2 - 1, n_1 k_1)$ wavelength interchangers. We show that this is optimal for the class of split cross-connects. However, we leave as an open question whether the family of split cross-connects are optimal if the number of input fibers is not equal to the number of output fibers and the set of supported input wavelengths differs from the set of supported output wavelengths.

2. Definitions.

2.1. Wavelength division multiplexing. WDM technology allows multiple signals to be routed along an optical fiber simultaneously by encoding each signal on its own wavelength. For example, a WDM network might allow each fiber to carry up to n signals simultaneously by giving each signal its own wavelength out of a set of n supported wavelengths. Devices crucial to such networks are *optical switches*. An optical switch has an arbitrary number of input fibers and output fibers, and any signal encoded on any wavelength on any of the input fibers can be routed to the same wavelength on any of the outgoing fibers. If these are the only devices included in an optical network, then a connection that is originally routed on a particular wavelength, say λ , must stay on that wavelength for its entire route. Even in simple networks this can result in congestion of the network even when no single fiber is carrying all n signals [6, 14]. Therefore it is often desirable to allow the wavelength of a connection to be changed. A *wavelength interchanger* is a device that has a single input fiber and a single output fiber. A wavelength interchanger can change the wavelength of any subset of the signals entering it provided the wavelengths that it encodes those signals on are unused on its output fiber; see Figure 1.

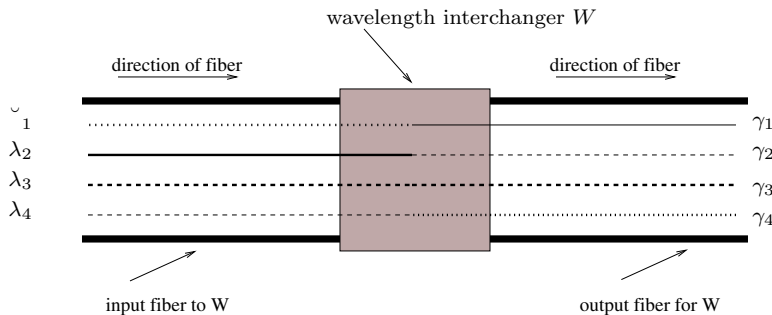


FIG. 1. A wavelength interchanger W with four signals passing through it. The signals on input wavelengths λ_1 , λ_2 , λ_3 , and λ_4 are changed to output wavelengths γ_1 , γ_2 , γ_3 , and γ_4 , respectively.

2.2. WDM cross-connects. In a WDM network, fibers join at various places, and signals on the incoming fibers can be routed to the outgoing fibers. The part of the network that allows this routing of signals from some set of incoming fibers to a set of outgoing fibers is referred to as a WDM *cross-connect*. A $k_1 \times k_2$ WDM cross-connect has k_1 input fibers and k_2 output fibers. We define $I = \{I_1, \dots, I_{k_1}\}$

to be the set of input fibers and $O = \{O_1, \dots, O_{k_2}\}$ to be the set of output fibers. The set of n_1 wavelengths, $\{\lambda_1, \dots, \lambda_{n_1}\}$, supported on each of the k_1 input fibers is referred to as the set of *input wavelengths* and similarly the set of *output wavelengths* is the set of n_2 wavelengths, $\{\gamma_1, \dots, \gamma_{n_2}\}$ supported on each of the output fibers. Without loss of generality we assume that $k_1 \leq k_2$ and $n_1, n_2 \geq 2$.

In general, a demand on a WDM cross-connect could be specified in various ways. For this paper we define a *demand* $d = (I_x, \lambda_y, O_z, \gamma_w)$ to be a request for a path and an assignment of wavelengths along the path from input fiber I_x to output fiber O_z such that the connection starts on input wavelength λ_y , changes wavelength only at a wavelength interchanger, and ends on output wavelength γ_w . Such a path and wavelength assignment is called a *route*. A routing R of D is a collection of routes, one for each demand $d \in D$, such that no two demands are assigned the same wavelength along a shared section of fiber. We will show that for some classes of cross-connects, this demand and route model is equivalent to models in which a demand does not specify the input and/or the output wavelength and a route is allowed to use any wavelengths available on the specified input and output fibers. We leave as an open question whether more flexible demand models allow for improved results for other types of cross-connects.

Given an existing set of demands, D , we say demand $d = (I_x, \lambda_y, O_z, \gamma_w)$ is *valid* with respect to D if input wavelength λ_y is unused on input fiber I_x by all demands in D and output wavelength γ_w is unused on output fiber O_z by all demands in D . A route r for d is *valid* with respect to a routing R of D if the wavelength assigned to d along each fiber in r is not used by any other connection currently routed on the fiber according to R .

2.3. Nonblocking properties. Ideally a cross-connect will always be able to satisfy a valid demand with a valid route. Depending on various assumptions, a number of different levels of *nonblocking* have been defined as follows.

- **Rearrangeably nonblocking:** A cross-connect is *rearrangeably nonblocking* if any new valid demand can be assigned a valid route provided that routes for already existing demands are allowed to be rerouted.
- **Wide-sense nonblocking:** A cross-connect is *wide-sense nonblocking* if any new valid demand can be assigned a route by an algorithm A provided that all currently routed demands were routed using A .
- **Strictly nonblocking:** A cross-connect is *strictly nonblocking* if any new valid demand can be assigned a route regardless of the routes assigned to any currently routed demands.

Strictly nonblocking and wide-sense nonblocking cross-connects have many advantages over rearrangeably nonblocking cross-connects. For example, a rearrangeably nonblocking cross-connect will create a buffering problem when it must interrupt currently routed connections to reroute them. While both strictly nonblocking and wide-sense nonblocking cross-connects avoid this buffering problem, a strictly nonblocking cross-connect is perhaps more robust since it does not depend upon having taken care to route previously requested demands in order to guarantee a route for any new demands. This is particularly desirable in practice because often hardware failures necessitate that some routes are chosen without regard to a particular algorithm. A wide-sense nonblocking cross-connect might require that all routes are rerouted after such a hardware failure. In contrast, since a strictly nonblocking cross-connect does not depend on the cross-connect having routed existed demands carefully, it is capable of handling situations where the set of existing routes may have been chosen to work

around a hardware failure without regard for how they utilize the resources inside the cross-connect. We focus our attention on the design of strictly nonblocking $k_1 \times k_2$ WDM cross-connects but note that the design of optimal wide-sense nonblocking cross-connects would also be of interest.

2.4. Topologies of cross-connects.

2.4.1. Split cross-connects. A *fabric* is defined to be a subnetwork that does not contain any wavelength interchangers. A $k_1 \times k_2$ WDM cross-connect is said to be a *split cross-connect* if it has k_1 input fibers connected to a fabric, F_1 , and k_2 output fibers connected to a second fabric, F_2 . Furthermore, the output fibers of F_1 are the input fibers for a set W of wavelength interchangers and the output fibers for the set W of wavelength interchangers are the input fibers of the fabric F_2 . Another way to say this is to say that C is a split cross-connect if every path from an input fiber to an output fiber through C passes through exactly one wavelength interchanger; see Figure 2.

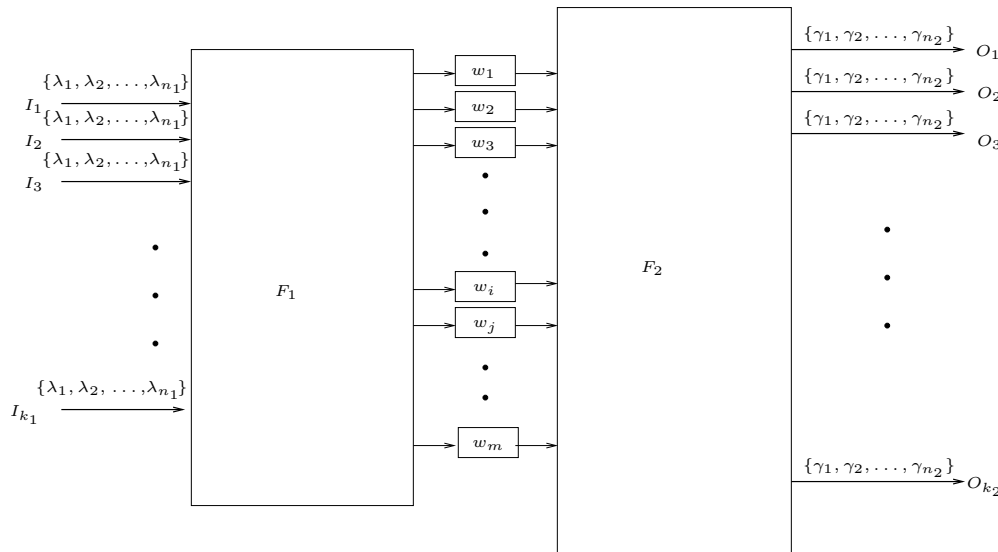


FIG. 2. A $k_1 \times k_2$ WDM split cross-connect with m wavelength interchangers. The set of k_1 input fibers is connected to the m wavelength interchangers with a fabric F_1 . The set of m wavelength interchangers is connected to the k_2 output fibers with a fabric F_2 .

The designs for strictly nonblocking cross-connects that we present are all split cross-connects. We will prove that in the case where the number of input and output fibers is the same, this is optimal. Note that without loss of generality we will overload the definition of split and allow it to also refer to any cross-connect for which every path p from any input fiber to any output fiber has *at most* one wavelength interchanger.

2.4.2. Homogeneous and heterogeneous cross-connects. We consider cross-connects for two different situations. The first situation involves cross-connects that have the same number of input fibers as output fibers. In these cross-connects the same set of input and output wavelengths are supported throughout the cross-connect. We call such cross-connects *homogeneous* cross-connects.

In the second case, the purpose of the cross-connect is to connect a set of fibers from a given network to a set of fibers from another network. In this case the set of wavelengths supported on the input fibers may differ from the set of wavelengths supported on the output fibers. Also the number of input fibers may differ from the number of output fibers. We call such a cross-connect a *heterogeneous* cross-connect.

3. Previous results. This work draws both from the literature of classical cross-connects and from more recent work on WDM cross-connects.

A classical architecture considers the case where each input and output line carries only one signal at a time. Designs for cross-connects in classical networks have been well studied [3, 12, 8, 1, 2, 7]. In a classical network the optimization criterion is the number of edges in the graph representing the topology of the cross-connect. In contrast, in wavelength interchanging WDM cross-connects the optimality criterion is the number of wavelength interchangers.

In general the cost of a design of a WDM network is dominated by the number of wavelength interchangers needed in the network. Thus the problem of optimizing WDM networks has been studied in a model that assigns a cost of one to any node in the network where wavelength conversion capability is provided [14, 6].

Other work has focused on the possibility of using less powerful (and therefore less expensive) wavelength conversion devices. One example would be a device that was capable of swapping two wavelengths while leaving the others fixed. That work has considered what effect using such less powerful wavelength interchangers has on the number of wavelength interchangers needed in total [9].

Recently some work has been done on considering the cost of providing complete wavelength conversion capability in a WDM cross-connect. For instance, various wavelength interchanging WDM homogeneous cross-connects with a variety of nonblocking capabilities have been considered and in particular one design that used $k \log k$ wavelength interchangers was shown to be strictly nonblocking [13]. We will improve upon this result by presenting a design for a strictly nonblocking cross-connect with $2k - 1$ wavelength interchangers.

4. Our contribution. We will consider homogeneous and heterogeneous cross-connects. We present the design of a strictly nonblocking $k_1 \times k_2$ WDM cross-connect that uses $k_1 + k_2 - 1$ wavelength interchangers. We also present the design of strictly nonblocking $k_1 \times k_2$ WDM cross-connects that use $n_1 k_1$ wavelength interchangers. Together these provide a family of strictly nonblocking $k_1 \times k_2$ WDM cross-connects that use at most $\min(k_1 + k_2 - 1, n_1 k_1)$ wavelength interchangers.

When considering the optimality of these designs we handle the homogeneous and heterogeneous cases separately. We start our discussion of the optimal number of wavelength interchangers in a homogeneous cross-connect by showing that if there exists a family of optimal nonsplit cross-connects, then there exists a family of split cross-connects that use the same number of wavelength interchangers. This reduces the problem to a special case of showing that a heterogeneous $k \times k$ WDM split cross-connect requires $2k - 1$ wavelength interchangers. For the heterogeneous case we consider only split cross-connects and leave as an open question whether there exist strictly nonblocking nonsplit cross-connects with strictly fewer wavelength interchangers. We present a sequence of demands that requires that any strictly nonblocking $k_1 \times k_2$ WDM heterogeneous split cross-connect use at least $\min(k_1 + k_2 - 1, n_1 k_1)$ wavelength interchangers. These results together show that the family of strictly nonblocking $k_1 \times k_2$ WDM cross-connects presented in section 5 are optimal for homogeneous cross-connects and heterogeneous split cross-connects. Finally, we consider

more relaxed demand models where the input and output wavelengths are not specified as part of the demand. We show that even under this model of demands, $2k - 1$ wavelength interchangers are necessary and sufficient for homogeneous cross-connects. Preliminary versions of much of this work were presented in [11, 10].

4.1. Overview. In section 5 we present the design of a family of $k_1 \times k_2$ WDM cross-connects. We show that these are strictly nonblocking. For the case of homogeneous cross-connects we prove, in section 6, that if there is a family of optimal nonsplit cross-connects, then there is a family of optimal split cross-connects with the same number of wavelength interchangers. We then restrict our attention to cross-connects with split designs. We present a sequence of demands that shows that any homogeneous strictly nonblocking $k \times k$ WDM cross-connect must have $2k - 1$ wavelength interchangers. We then show that the lower bounds presented hold under demand models that do not specify the input and output wavelength as part of the demand. For the heterogeneous case we show, in section 7, that any strictly nonblocking $k_1 \times k_2$ WDM split cross-connect must have $\min(k_1 + k_2 - 1, n_1 k_1)$ wavelength interchangers. Finally, in section 8, we consider future work.

5. Designs for a family of strictly nonblocking cross-connects.

5.1. Traditional cross-connects. We start our discussion with a look at traditional cross-connect designs and how they apply to the design of WDM cross-connects. In a classical network, a *line* carried only one *channel* (wavelength) and therefore no two signals shared a line. A traditional strictly nonblocking $k_1 \times k_2$ cross-connect had k_1 input lines and k_2 output lines. A demand was a request for a connection from an input line to an output line. Note that in this setting at most one demand could use any particular input or output line at a given time. A route for a demand in a classical cross-connect was a path from the input line to the output line that was edge-disjoint from any existing route in the cross-connect. Therefore we say that the topology of a strictly nonblocking classical cross-connect is pathwise strictly nonblocking.

In the WDM setting we require that no two signals on the same wavelength share a fiber. Therefore if a WDM fabric has no wavelength conversion capability, then a demand will be a request for a connection on a specified wavelength from an input fiber to an output fiber. A route for such a demand is a path from the input fiber to the output fiber that is edge-disjoint (fiber-disjoint) from all other existing demands on the same wavelength. Therefore it is natural to extend the designs of classical cross-connects to WDM fabrics as follows.

Suppose we consider the topology of a classical cross-connect C . The topology of C can be represented by a graph, G , where the edges correspond to lines in C and the nodes correspond to components in C . We create a WDM fabric F from G by putting a WDM fiber in F , supporting n wavelengths, for each edge in G and a WDM switch in F for each node in G . Notice that since we have not placed any wavelength interchangers in F , a signal passing through F must stay on the same wavelength for its entire path through F . Furthermore, since signals on different wavelengths do not interfere with each other in WDM fabrics, one can conceptually think of F as allowing us to have n copies of C , one for each of the n wavelengths. Therefore if C was pathwise strictly nonblocking, then for each wavelength λ_i we are guaranteed that we can route a demand from input fiber a to output fiber b on wavelength λ_i as long as both a and b currently do not have a demand that uses wavelength λ_i .

A WDM fabric that is pathwise strictly nonblocking greatly simplifies the design of a strictly nonblocking WDM cross-connect and therefore we will make use of this

idea when presenting our designs.

5.2. Split cross-connects. We start with the design of a strictly nonblocking $k_1 \times k_2$ WDM split cross-connect. First, we connect the input fibers to a set of $k_1 + k_2 - 1$ wavelength interchangers with a fabric F_1 . The only restriction on F_1 is that it must have the topology of a pathwise strictly nonblocking $k_1 \times (k_1 + k_2 - 1)$ classical cross-connect. We then connect the set of $k_1 + k_2 - 1$ wavelength interchangers to the input side of another fabric F_2 . The output side of F_2 is the set of k_2 output fibers of our cross-connect. The only restriction on F_2 is that it has the topology of a pathwise strictly nonblocking $(k_1 + k_2 - 1) \times k_2$ classical cross-connect. This family of strictly nonblocking $k_1 \times k_2$ WDM cross-connects is by definition a family of split cross-connects; see Figure 3.

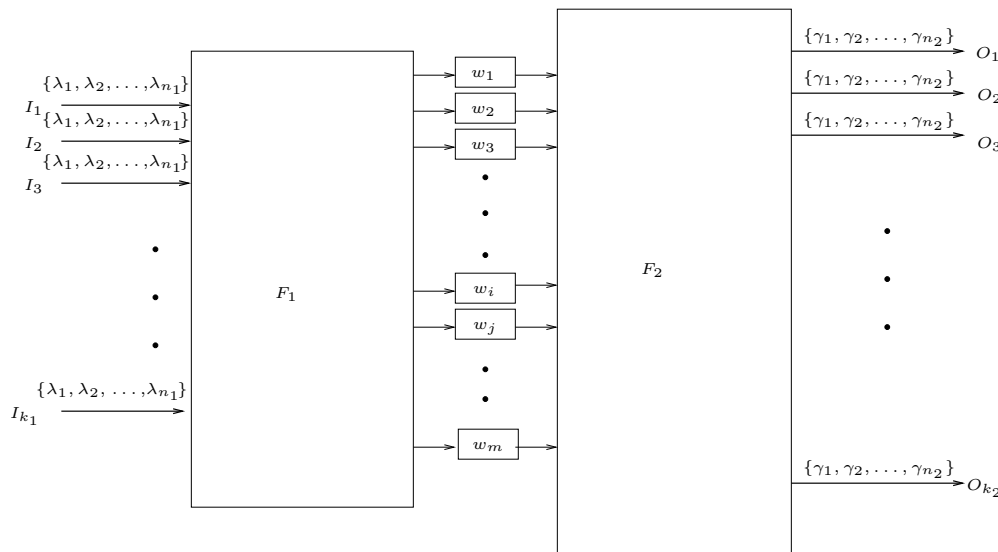


FIG. 3. A $k_1 \times k_2$ WDM split cross-connect with $m = k_1 + k_2 - 1$ wavelength interchangers. The set of k_1 input fibers is connected to the m wavelength interchangers with a fabric F_1 . The set of m wavelength interchangers is connected to the k_2 output fibers with a fabric F_2 .

THEOREM 1. Let C be a $k_1 \times k_2$ WDM split cross-connect where the number of wavelength interchangers is $k_1 + k_2 - 1$. Suppose that the topology of F_1 is that of some pathwise strictly nonblocking $k_1 \times (k_1 + k_2 - 1)$ cross-connect. Similarly, F_2 has the topology of a pathwise strictly nonblocking $(k_1 + k_2 - 1) \times k_2$ cross-connect. Then C is a strictly nonblocking $k_1 \times k_2$ WDM cross-connect.

Proof. To show that C is strictly nonblocking it is enough to show that for any existing set of demands D that are currently routed through C and for any new valid demand $d = (a, \lambda_1, b, \gamma_2)$ where $D \cup \{d\}$ is a valid demand set, there is a valid route for d . Let R be any valid routing of D .

We will break D into three sets of demands. First, let D_1 be the set of demands in D that use λ_1 as their input wavelength and let D_2 be the set of demands in D that use γ_2 as their output wavelength. Finally, let D_3 be $D \setminus (D_1 \cup D_2)$. Notice that no demands in D_3 can possibly keep us from routing d since these demands all use a wavelength other than λ_1 as they pass through F_1 and a wavelength other than γ_2 as they pass through F_2 . Therefore we restrict our attention to showing that d cannot be blocked by demands in $D_1 \cup D_2$.

Let W be the set of wavelength interchangers without a demand in D_1 or D_2 routed through them. First, we show that W is nonempty. Since d is a valid demand, input fiber a could not have a demand in D that used input wavelength λ_1 and output fiber b could not have a demand in D that used output wavelength γ_2 . Hence, $|D_1| \leq k_1 - 1$ and $|D_2| \leq k_2 - 1$ and therefore $|D_1 \cup D_2| \leq k_1 + k_2 - 2$. Since C is a split cross-connect, each route passes through at most one wavelength interchanger. Therefore $|W| \geq k_1 + k_2 - 1 - |D_1 \cup D_2| \geq 1$.

Let $w \in W$ be a wavelength interchanger that does not currently have a demand routed through it using either λ_1 or γ_2 . If there exists a route from a to w and a route from w to b , then there exists a route for d through C . However, since F_1 has the topology of a pathwise strictly nonblocking $k_1 \times (k_1 + k_2 - 1)$ cross-connect, there must exist a path from a to w that is edge-disjoint from all other paths for demands in D_1 . Similarly, there must exist a path in F_2 from w to b that is edge-disjoint from all other paths through F_2 for demands in D_2 .

Therefore there exists a valid route through C for d and as a result C is strictly nonblocking. \square

We call this type of strictly nonblocking $k_1 \times k_2$ WDM cross-connect a *standard* strictly nonblocking $k_1 \times k_2$ WDM cross-connect.

5.3. Dedicated cross-connects. Note that the family of standard $k_1 \times k_2$ WDM cross-connects presented in the previous section uses $k_1 + k_2 - 1$ wavelength interchangers. For homogeneous cross-connects we will show in section 6 that this is optimal. However, for heterogeneous cross-connects we must also consider a limiting case.

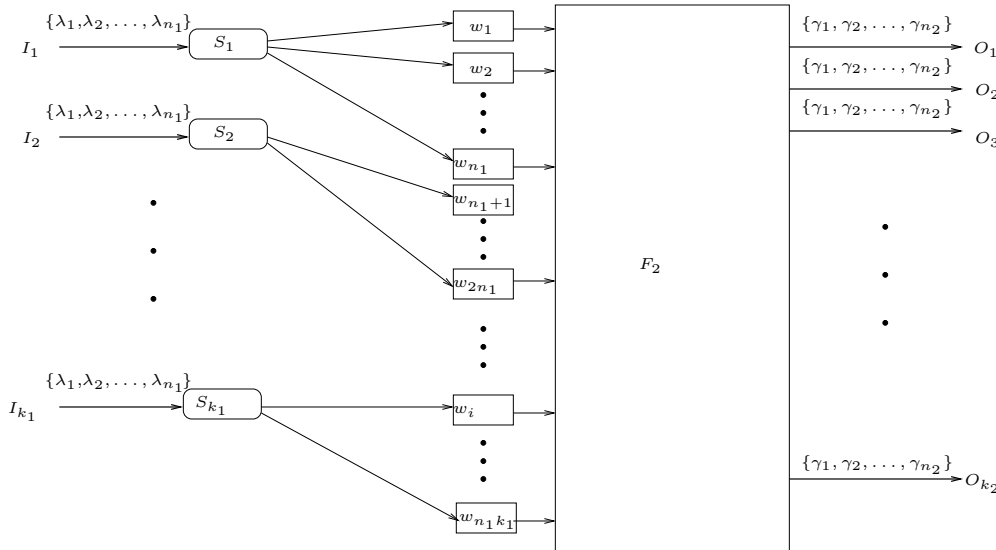


FIG. 4. A $k_1 \times k_2$ WDM dedicated cross-connect with $n_1 k_1$ wavelength interchangers. For each of the k_1 input fibers, I_i , there is a $1 \times n_1$ switch S_i . The n_1 output fibers of S_i are connected to n_1 wavelength interchangers that are used only for demands from I_i . The set of $n_1 k_1$ wavelength interchangers is connected to the k_2 output fibers with a fabric F_2 .

Notice that for a heterogeneous $k_1 \times k_2$ WDM cross-connect with n_1 supported input wavelengths on each input fiber, the maximum number of demands in any valid demand set is $n_1 k_1$. In particular, if $k_2 > (n_1 - 1)k_1$, then the maximum number of

possible demands is no more than $k_1 + k_2 - 1$. In this case we will use what we refer to as a dedicated cross-connect.

In a *dedicated* $k_1 \times k_2$ WDM cross-connect each input fiber is connected to its own $1 \times n_1$ optical switch and the n_1 output fibers of this switch are then connected to n_1 wavelength interchangers. Conceptually, each input fiber has n_1 wavelength interchangers dedicated to service demands from it. The set of $n_1 k_1$ wavelength interchangers is connected to the set of output fibers with a fabric F_2 that has the topology of a pathwise strictly nonblocking $(n_1 k_1) \times k_2$ cross-connect; see Figure 4.

THEOREM 2. *Let C be a $k_1 \times k_2$ WDM dedicated cross-connect where the number of wavelength interchangers is $n_1 k_1$. Then C is strictly nonblocking.*

Proof. Each of the k_1 input fibers can support at most n_1 demands. For each input fiber there are n_1 wavelength interchangers that are reachable from that input fiber alone. Therefore, each wavelength interchanger can be reserved for a demand using a specific input wavelength on the input fiber that can reach that wavelength interchanger. The fabric F_2 is pathwise strictly nonblocking. Therefore, after the demand has been routed through the wavelength interchanger, it can be routed through F_2 to the appropriate output fiber. \square

5.4. Minimal strictly nonblocking cross-connects. By choosing either a dedicated or standard cross-connect, this set of designs provides a family of $k_1 \times k_2$ WDM cross-connects that use $\min(k_1 + k_2 - 1, n_1 k_1)$ wavelength interchangers. In the next section we prove that this is optimal for homogeneous cross-connects and heterogeneous split cross-connects.

6. Lower bound for homogeneous cross-connects. In section 5 we presented a family of strictly nonblocking $k_1 \times k_2$ WDM cross-connects that contain $\min(k_1 + k_2 - 1, n_1 k_1)$ wavelength interchangers. If the cross-connect is a homogeneous cross-connect with k input fibers and k output fibers, these designs use $2k - 1$ wavelength interchangers. In this section we present matching lower bounds. In general, the flavor of these proofs is that we assume that we have a strictly nonblocking $k \times k$ WDM cross-connect that uses $m < 2k - 1$ wavelength interchangers and show that there is a sequence of demands and a routing of those demands such that there is a final valid demand d that cannot be routed by the cross-connect.

Although we presented our upper bounds for homogeneous and heterogeneous cases simultaneously, in this section we handle only the homogeneous case and assume that $k = k_1 = k_2$, $n = n_1 = n_2$ and the set of input wavelengths, $\{\lambda_1, \dots, \lambda_n\}$, is also the set of output wavelengths. We leave the heterogeneous case for the next section.

Initially, we prove our lower bounds using a demand model where the input and output wavelengths are specified as part of the demand. In this setting we prove that if there is some $k \geq 3$ for which there is a strictly nonblocking $k \times k$ WDM cross-connect with $m < 2k - 1$ wavelength interchangers, then there exists some $k' \geq 2$ for which there is a strictly nonblocking $k' \times k'$ WDM *split* cross-connect with $m' < 2k' - 1$ wavelength interchangers. We then show that any strictly nonblocking $k \times k$ WDM *split* cross-connect requires $2k - 1$ wavelength interchangers. Combining these results shows that any strictly nonblocking $k \times k$ WDM cross-connect requires $2k - 1$ wavelength interchangers.

We then consider a demand model in which a demand consists of only an input and output fiber and the cross-connect is allowed to choose the input and output wavelengths from the set of available input and output wavelengths. We present a simple construction that extends the proofs in this section to hold in this new demand model.

6.1. The adversarial view of a strictly nonblocking cross-connect. In this section we will prove a lower bound on the number of wavelength interchangers necessary in any strictly nonblocking $k \times k$ WDM cross-connect. To prove such a lower bound, we will present a sequence of demands and a sequence of valid routings the cross-connect could have chosen that eventually force the cross-connect to use at least $2k - 1$ wavelength interchangers.

We assume that we are given a strictly nonblocking $k \times k$ WDM cross-connect C . As an exercise, first suppose that C is such that there is an edge disjoint path from every input fiber to every wavelength interchanger and an edge disjoint path from every wavelength interchanger to every output fiber. Suppose C contains $2k - 2$ wavelength interchangers. For the purposes of this example, consider a demand set with $k - 1$ demands with input wavelength λ_1 and output wavelength λ_1 and $k - 1$ demands with input wavelength λ_2 and output wavelength λ_2 . A valid routing of these demands would use $k - 1$ wavelength interchangers to route the demands with input wavelength λ_1 and output wavelength λ_1 . Furthermore, it *could* route the $k - 1$ demands with input wavelength λ_2 and output wavelength λ_2 through a disjoint set of $k - 1$ wavelength interchangers. By the definition of strictly nonblocking, it must be possible to route another demand through C even if this is the current state of C . However, a new demand with input wavelength λ_1 and output wavelength λ_2 cannot be serviced by any of the wavelength interchangers in C . Thus C cannot be strictly nonblocking.

In general, we cannot assume that C has this structure. Therefore we will play the part of the adversary that is attempting to force C to reveal $2k - 1$ wavelength interchangers. Initially, we do not know anything about the structure of the cross-connect. Each time we place a demand d on the cross-connect, the route r that C uses to satisfy d becomes known to us. Suppose that at some point there is a set of demands D on the cross-connect and a routing R of D . Furthermore, let d be a demand that is valid with respect to D . If there is a route r that could be used to satisfy d and r is known to us at that point, then we can also insist that the cross-connect uses r to route d .

Therefore when proving our lower bounds we will sometimes insist that a demand be routed according to a particular route that is already known to exist.

6.2. Existence of minimum cost split cross-connects. In this section we prove that for the case of homogeneous $k \times k$ WDM cross-connects it is sufficient to consider only homogeneous $k \times k$ WDM *split* cross-connects. We reduce our problem to considering only strictly nonblocking $k \times k$ WDM split cross-connects through the following steps.

1. We define the set *Long* of strictly nonblocking cross-connects that contain at least one path with multiple wavelength interchangers along the path.
2. By definition, *Long* cannot contain a strictly nonblocking 1×1 WDM cross-connect with 1 wavelength interchanger. This provides a base case for our inductive proof.
3. We then consider the smallest $k > 2$ such that there exists a strictly nonblocking $k \times k$ WDM cross-connect $C \in \text{Long}$ with fewer than $2k - 1$ wavelength interchangers and show that C can be reduced to a smaller strictly nonblocking $k' \times k'$ WDM cross-connect C' with the property that C' has fewer than $2k' - 1$ wavelength interchangers.
4. Thus, 2 and 3 together imply that if there exists a strictly nonblocking $k \times k$ WDM cross-connect $C \in \text{Long}$ with fewer than $2k - 1$ wavelength inter-

changers, then we can reduce it to a strictly nonblocking $k' \times k'$ WDM cross-connect $C' \notin Long$ with fewer than $2k' - 1$ wavelength interchangers.

More precisely, we define *Long* to be the set of strictly nonblocking WDM cross-connects that contain at least one directed path P from some input node $a \in I$ through $w_P > 1$ wavelength interchangers to some output node $b \in O$. We say P is a *long* path; see Figure 5.

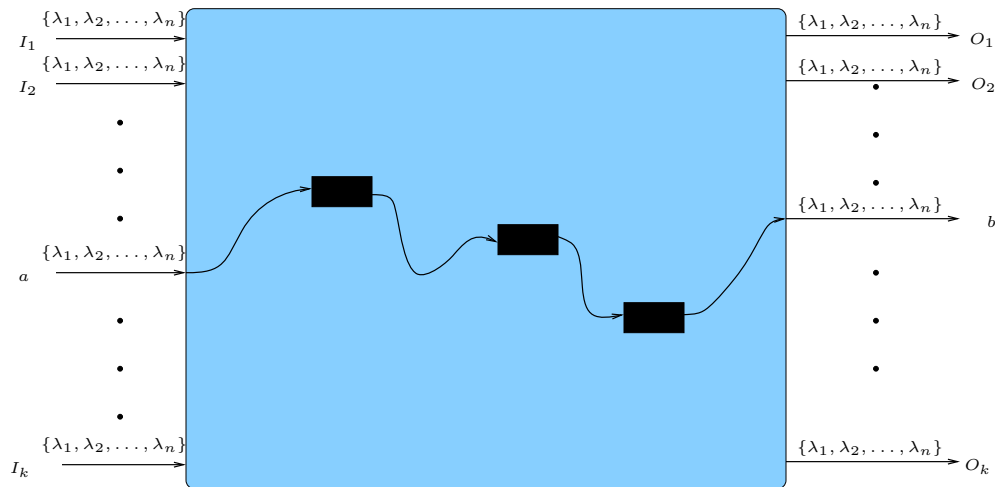


FIG. 5. A long path in C with three wavelength interchangers.

To reduce a cross-connect $C \in Long$ to a smaller strictly nonblocking cross-connect, we make n demands from the input fiber a to the output fiber b . By assumption, there exists a long path P from a to b . Therefore we can insist that C route these n demands along P . As long as these demands are routed along P , no other demands on C can use any part of P . Therefore all future demands will be handled by other parts of C . Thus, C is then equivalent to a smaller cross-connect that has one less input fiber and one less output fiber and at least two fewer wavelength interchangers.

Formally, for any $C \in Long$ with n wavelengths supported on each input and output fiber and any long path $P \in C$ from input fiber a to output fiber b , we define the operation $\mathbf{Fill}(C, P, n)$ that routes demands $(a, \lambda_i, b, \lambda_i)$ for all $1 \leq i \leq n$ along P with constant wavelength assignment λ_i .

LEMMA 3. *For any $C \in Long$, the result of the operation $\mathbf{Fill}(C, P, n)$ is that no new demands can be routed through any wavelength interchanger on P .*

Proof. Let D be the set of demands made by the operation $\mathbf{Fill}(C, P, n)$. Since $\mathbf{Fill}(C, P, n)$ routes one demand per wavelength, with constant wavelength assignment, along P , all wavelengths on all fibers in P are used for demands in D . Therefore there are no available supported wavelengths for any input or output fibers for any wavelength interchangers on P . \square

By definition, *Long* cannot contain a strictly nonblocking 1×1 WDM cross-connect with a 1 wavelength interchanger. Thus we now focus our attention on cross-connects in *Long* with $k \geq 2$. We show that we can reduce a strictly nonblocking $k \times k$ WDM cross-connect $C \in Long$ with $m < 2k - 1$ wavelength interchangers to a strictly nonblocking $k' \times k'$ WDM cross-connect with $m' < 2k' - 1$ wavelength interchangers.

THEOREM 4. *If for some $k \geq 2$, there exists a $k \times k$ WDM cross-connect $C \in Long$ that has fewer than $2k - 1$ wavelength interchangers, then for some k' , where $1 \leq k' < k$, there exists a strictly nonblocking $k' \times k'$ WDM cross-connect $C' \notin Long$ that has fewer than $2k' - 1$ wavelength interchangers.*

Proof. By the definition of *Long*, if $C \in Long$ then $k \geq 2$. Let $k \geq 2$ be the smallest number such that there exists some $k \times k$ WDM cross-connect $C \in Long$ with $m < 2k - 1$ wavelength interchangers. Let w_P be the number of wavelength interchangers on a long path P from input fiber a to output fiber b that must exist in C by the definition of the class *Long* of cross-connects. Let n be the number of supported wavelengths.

We will use $\mathbf{Fill}(C, P, n)$ to route n demands along P . Once these demands have been routed along P , no new demands can use any portion of P . Thus any new demand on C must be routed through other sections of C . Therefore we show that C , with these n demands routed along P , is equivalent to a strictly nonblocking $k' \times k'$ WDM cross-connect C' with one less input fiber, one less output fiber, and w_p fewer wavelength interchangers.

We start by performing $\mathbf{Fill}(C, P, n)$ and assume that the demands placed on C by $\mathbf{Fill}(C, P, n)$ are not removed. By Lemma 3 no new demands can be routed through wavelength interchangers on P . Thus consider the cross-connect C' obtained by the process $\mathbf{Modify}(C, P)$ that is defined as follows. Remove input fiber a and output fiber b from C . Remove all fibers along path P . All wavelength interchangers along P are isolated (i.e., have no incoming or outgoing fibers) and so they are also removed. This construction can easily be seen to have the property that after performing $\mathbf{Fill}(C, P, n)$ any other demand will have a routing and wavelength assignment in C if and only if it does in C' . See Figure 6 to see the change to C as a result of performing $\mathbf{Fill}(C, P, n)$ and $\mathbf{Modify}(C, P)$.

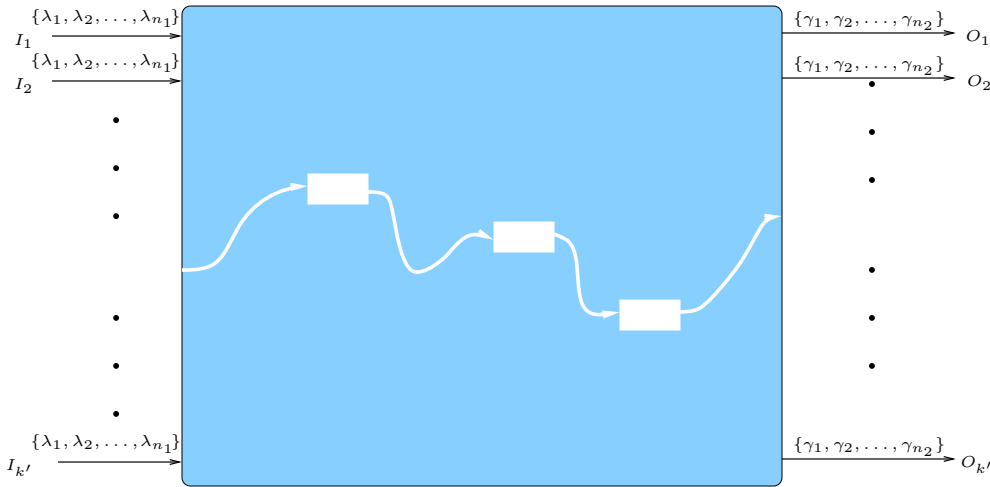


FIG. 6. The result of performing $\mathbf{Fill}(C, P, n)$ and $\mathbf{Modify}(C, P)$.

Therefore since C is strictly nonblocking, C' must also be strictly nonblocking. Notice that since we have removed one input and output fiber from C to create C' , C' is a $k' \times k'$ WDM cross-connect where $k' = k - 1$. Furthermore, we removed at least two wavelength interchangers from C in $\mathbf{Modify}(C, P)$ and therefore the number of wavelength interchangers is $m - w_P < 2k' - 1$. Therefore C' is a $k' \times k'$ WDM cross-

connect with fewer than $2k' - 1$ wavelength interchangers. Since k was assumed to be the smallest number such that there exists a $k \times k$ WDM cross-connect $C \in Long$ with fewer than $2k - 1$ wavelength interchangers and since $k' < k$ and $m' < 2k' - 1$, $C' \notin Long$. Thus $C' \notin Long$ is a strictly nonblocking $k' \times k'$ WDM cross-connect with fewer than $2k' - 1$ wavelength interchangers. \square

Theorem 4 says that if for some $k \geq 2$ there is an optimal strictly nonblocking $k \times k$ WDM cross-connect C with $m < 2k - 1$ wavelength interchangers and C is not a split cross-connect, then there is some $k' \geq 1$ for which there is a strictly nonblocking $k' \times k'$ WDM cross-connect C' with $m' < 2k' - 1$ wavelength interchangers. Furthermore, C' must be a split cross-connect. In the next section we will show that any strictly nonblocking $k \times k$ WDM split cross-connect must have at least $2k - 1$ wavelength interchangers.

6.3. Lower bound for split cross-connects. In the last section we reduced the problem of showing that the optimal number of wavelength interchangers for any strictly nonblocking $k \times k$ WDM cross-connect is $2k - 1$ to showing that this is true for any strictly nonblocking $k \times k$ WDM split cross-connect. We now show that any strictly nonblocking $k \times k$ WDM split cross-connect must have $2k - 1$ wavelength interchangers.

To do this we consider any strictly nonblocking WDM split cross-connect C with fewer than $2k - 1$ wavelength interchangers. For such a cross-connect we will create a demand set D and a routing R of D that routes D in such a way that C must use each wavelength interchanger to service a demand with either input wavelength λ_1 or output wavelength λ_2 . Given this set of demands D and routing R we will then show that one additional valid demand $d \notin D$ exists with input wavelength λ_1 and output wavelength λ_2 . Since all the wavelength interchangers in the cross-connect will already be servicing a demand that either uses this new demand's input wavelength or its output wavelength, the cross-connect will not have a wavelength interchanger to service the new demand. Thus any strictly nonblocking $k \times k$ WDM split cross-connect must have at least $2k - 1$ wavelength interchangers.

We say that a valid demand set D is *standard* if and only if

1. $|D| = 2k$ and
2. all of the demands in D use input wavelength λ_1 or λ_2 and output wavelength λ_1 or λ_2 .

Suppose we have a routing R_i for a standard set of demands D_i . Let W_i^B be the set of wavelength interchangers that service a demand with either input wavelength λ_1 or output wavelength λ_2 . These wavelength interchangers are thought of as "blocking" demands with input wavelength λ_1 and output wavelength λ_2 from being routed through them. Let W_i^F be the set of all other wavelength interchangers. The wavelength interchangers in W_i^F are thought to be "free" to service demands with input wavelength λ_1 and output wavelength λ_2 .

If a routing R_i of a standard set of demands D_i uses fewer than $2k - 1$ wavelength interchangers, then there must be at least two wavelength interchangers that are both in W_i^B and are each servicing two demands.

As an example, assume WI_u and WI_v are two such wavelength interchangers. For this example, suppose $(I_{u1}, \lambda_1, O_{u1}, \lambda_1)$ and $(I_{u2}, \lambda_2, O_{u2}, \lambda_2)$ are the two demands that WI_u services. Furthermore, suppose $(I_{v1}, \lambda_1, O_{v1}, \lambda_1)$ and $(I_{v2}, \lambda_2, O_{v2}, \lambda_2)$ are the two demands that WI_v services; see Figure 7.

Consider the following change to D_i .

- Remove $(I_{u1}, \lambda_1, O_{u1}, \lambda_1)$ and $(I_{v2}, \lambda_2, O_{v2}, \lambda_2)$ from D_i ; see Figure 8.

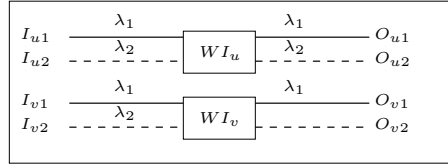


FIG. 7. Two wavelength interchangers that each service two demands.

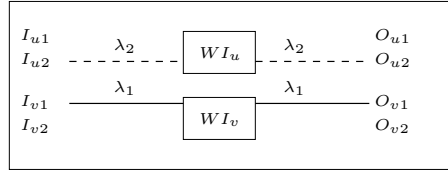


FIG. 8. The effect of removing demands $(I_{u1}, \lambda_1, O_{u1}, \lambda_1)$ and $(I_{v2}, \lambda_2, O_{v2}, \lambda_2)$.

Notice that WI_u and WI_v both remain in W_i^B after we remove these two demands. However, I_{u1} now can ask for a new demand with input wavelength λ_1 and O_{v2} can now ask for a new demand with output wavelength λ_2 . Now, we do the following.

- Add demand $(I_{u1}, \lambda_1, O_{v2}, \lambda_2)$ to D_i to create D_{i+1} ; see Figure 9.

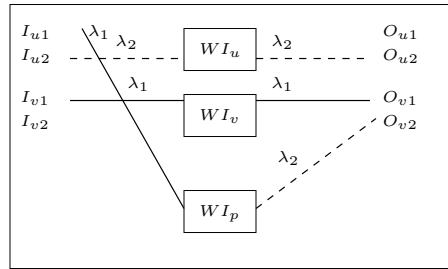


FIG. 9. The effect of adding demand $(I_{u1}, \lambda_1, O_{v2}, \lambda_2)$.

This new demand is a valid demand with respect to D_i . However, since it has input wavelength λ_1 and output wavelength λ_2 , it will not be able to be routed by any wavelength interchanger in W_i^B . Thus the route for this demand will force a new wavelength interchanger to service a demand with input wavelength λ_1 and output wavelength λ_2 . The wavelength interchanger that services this demand will thus be added to W_{i+1}^B . By repeating this process we eventually will show that either C is not strictly nonblocking or there exists a set of standard demands D_t and a routing of these demands such that $|W_t^B| \geq 2k - 1$ and thus C contains at least $2k - 1$ wavelength interchangers.

We now formalize this construction. First, we show that a standard set of demands D_0 exists and can be routed by C . Create k demands of the form $d_{i1} = (I_i, \lambda_1, O_i, \lambda_2)$ for $1 \leq i \leq k$. Clearly, since C is strictly nonblocking, C must be able to satisfy these k demands. For $1 \leq i \leq k$, create a demand $d_{i2} = (I_i, \lambda_2, O_i, \lambda_1)$ and route along the same path as d_{i1} . Notice that there are $2k$ demands and each demand uses input wavelength λ_1 or λ_2 and output wavelength λ_1 or λ_2 . Therefore this set of demands meets the definition of a standard set of demands.

Now we prove that any routing R_i of a standard set of demands D_i that uses

$2k - g$ wavelength interchangers will have g wavelength interchangers servicing two demands. Since a standard set of demands D consists of $2k$ demands each involving only wavelengths λ_1 and λ_2 , routing D will require at least k wavelength interchangers. Thus we assume that any cross-connect under consideration has at least k wavelength interchangers. Therefore in what follows, we always assume that $g \leq k$.

LEMMA 5. *If C is a strictly nonblocking $k \times k$ WDM split cross-connect, D_i is a standard set of demands and R_i is a routing of the demands in D_i that uses $2k - g$ wavelength interchangers, where $g > 0$, then g wavelength interchangers in W_i^B will each service two demands both of whose input wavelengths are λ_1 and λ_2 and whose output wavelengths are λ_1 and λ_2 .*

Proof. The $2k$ demands in D_i use only input wavelengths λ_1 and λ_2 and output wavelengths λ_1 and λ_2 . As a result no wavelength interchanger can service more than two of these demands. This implies that g wavelength interchangers in W_i^B or W_i^F must service two demands. Any such wavelength interchanger that services two of these demands must service a demand with input wavelength λ_1 and a demand with output wavelength λ_2 and therefore is by definition in W_i^B . \square

Given the standard set of demands D_0 and the routing R_0 of D_0 we now present two manipulations that together can be used to iteratively change the set of demands on C so that eventually we arrive at a standard set of demands D_t and a standard routing R_t of D_t such that every wavelength interchanger in C is servicing a demand with either input wavelength λ_1 or output wavelength λ_2 . Notice that this is equivalent to showing that if C has no more than $2k - g$ wavelength interchangers for some $g > 0$, then a standard set of demands D_i and a standard routing R_i of D_i exists such that $|W_i^B| = 2k - g$ and $|W_i^F| = 0$.

Let $WI_j \in W_i^B$ be a wavelength interchanger that services exactly two demands, d_1 and d_2 . Suppose these demands have the form $d_1 = (I_1, \lambda_1, O_2, \lambda_2)$ and $d_2 = (I_2, \lambda_2, O_1, \lambda_1)$. The operation **Uncross**(WI_j) is defined to have the effect of changing these two demands to be $(I_1, \lambda_1, O_1, \lambda_1)$ and $(I_2, \lambda_2, O_2, \lambda_2)$ and routing these new demands exactly as d_1 and d_2 were routed while keeping all other demands and routes unchanged. Recall that we can force the cross-connect to use a known route for a new demand in order to construct a particular routing. Notice that the only real change made by **Uncross**(WI_j) is to swap the output paths of the routes of d_1 and d_2 . On the other hand, if the demands have the form $d_1 = (I_1, \lambda_1, O_2, \lambda_1)$ and $d_2 = (I_2, \lambda_2, O_1, \lambda_2)$, then **Uncross**(WI_j) has no effect; see Figure 10.

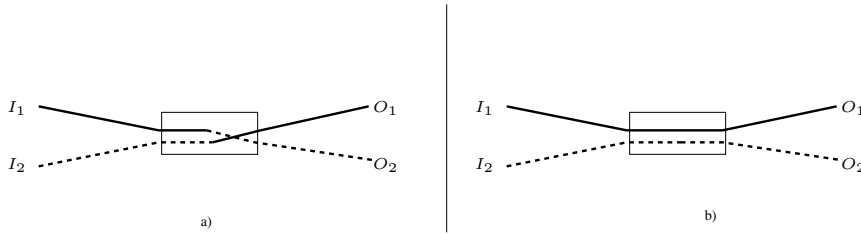


FIG. 10. *In a) the wavelength interchanger is shown before the operation **Uncross** is performed. The result of **Uncross** is shown in b).*

Next, we define the operation **Block**($C, (D_i, R_i)$) where C is a strictly nonblocking $k \times k$ WDM split cross-connect, D_i is a standard set of demands, and R_i is a routing of the demands in D_i that uses fewer than $2k - 1$ wavelength interchangers. The goal of **Block**($C, (D_i, R_i)$) is to alter D_i and R_i to create a new standard set of

demands D_{i+1} and a routing R_{i+1} for D_{i+1} such that $W_{i+1}^B > W_i^B$. In order to achieve this, $\mathbf{Block}(C, (D_i, R_i))$ will use the demands serviced by two of the wavelength interchangers in W_i^B that must be servicing two demands by Lemma 5 to create a new demand with input wavelength λ_1 and output wavelength λ_2 that must be serviced by a wavelength interchanger that was not in W_i^B . Figure 11 illustrates the steps of $\mathbf{Block}(C, (D_i, R_i))$.

$\mathbf{Block}(C, (D_i, R_i))$.

1. Take two wavelength interchangers, WI_u and WI_v , in W_i^B that, by Lemma 5, each service two demands.
2. $\mathbf{Uncross}(WI_u)$ and $\mathbf{Uncross}(WI_v)$.
3. Let $(I_{u1}, \lambda_1, O_{u1}, \lambda_1)$ and $(I_{u2}, \lambda_2, O_{u2}, \lambda_2)$ be the two resulting demands that WI_u services. Let $(I_{v1}, \lambda_1, O_{v1}, \lambda_1)$ and $(I_{v2}, \lambda_2, O_{v2}, \lambda_2)$ be the two resulting demands that WI_v services.
4. Remove $(I_{u1}, \lambda_1, O_{u1}, \lambda_1)$ and $(I_{v2}, \lambda_2, O_{v2}, \lambda_2)$ from D_i and route all remaining demands according to R_i to create D_{i+1} and R_{i+1} .
5. Add $(I_{v2}, \lambda_2, O_{u1}, \lambda_1)$ to D_{i+1} and add the route chosen by C for this demand to R_{i+1} .
6. Add $(I_{u1}, \lambda_1, O_{v2}, \lambda_2)$ to D_{i+1} and add the route chosen by C for this demand to R_{i+1} .
7. Return (D_{i+1}, R_{i+1}) .

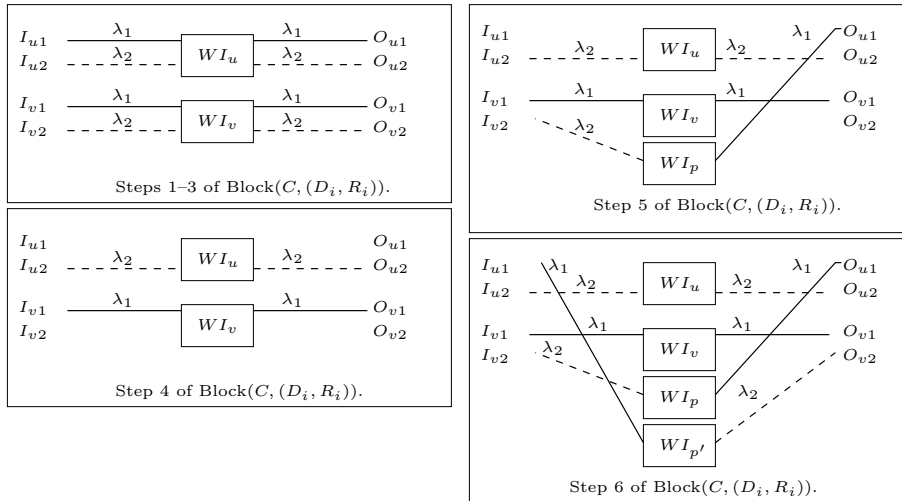


FIG. 11. The steps in $\mathbf{Block}(C, (D_i, R_i))$.

We start with D_0 and R_0 as defined above and then inductively define $(D_{i+1}, R_{i+1}) = \mathbf{Block}(C, (D_i, R_i))$ as long as R_i uses fewer than $2k - 1$ wavelength interchangers. Consider $\mathbf{Block}(C, (D_i, R_i))$. By Lemma 5 and the assumption that R_i uses fewer than $2k - 1$ wavelength interchangers, there must exist two wavelength interchangers in W_i^B that service two demands. Given these two wavelength interchangers, we then “uncross” their demands in Step 2. This does not change any of the wavelengths used along any of the fibers in C . In particular, while this alters the set of demands, the new set of demands is still a standard set of demands. In Step 4 we remove two demands and leave the remaining demands unchanged. By our choice of demands to remove we ensure that WI_v and WI_u will still each service one demand with either

input wavelength λ_1 or output wavelength λ_2 . Thus WI_v and WI_u remain in W_i^B .

Since C is strictly nonblocking, a valid route for the new demand in Step 6 must exist. A wavelength interchanger in W_i^F must be used to service this demand since all wavelength interchangers in W_i^B are servicing a demand with either input wavelength λ_1 or output wavelength λ_2 . Since the demand requested in Step 6 has both input wavelength λ_1 and output wavelength λ_2 , the wavelength interchanger that services this new demand will be in W_{i+1}^B . Notice that the demand in Step 5 ensures that there are a total of $2k$ demands on C . Hence, after Step 6, D_{i+1} is a standard set of demands and R_{i+1} is a routing for those demands such that $|W_{i+1}^B| = |W_i^B| + 1$ and $|W_{i+1}^F| = |W_i^F| - 1$. Thus we can conclude the following lemma.

LEMMA 6. *If C is a strictly nonblocking $k \times k$ WDM split cross-connect, D_i is a standard set of demands and R_i is a routing of the demands in D_i that uses fewer than $2k - 1$ wavelength interchangers, then $\mathbf{Block}(C, (D_i, R_i))$ can be executed and $(D_{i+1}, R_{i+1}) = \mathbf{Block}(C, (D_i, R_i))$, where D_{i+1} is a standard set of demands and R_{i+1} is a routing of the demands in D_{i+1} . Also, D_{i+1} and R_{i+1} are such that $|W_{i+1}^B| = |W_i^B| + 1$ and $|W_{i+1}^F| = |W_i^F| - 1$.*

We now use Lemma 6 to show that there must be $2k - 1$ wavelength interchangers in any strictly nonblocking $k \times k$ WDM split cross-connect.

THEOREM 7. *For any strictly nonblocking $k \times k$ WDM split cross-connect there must be a standard set of demands D_t and a standard routing R_t of D_t that uses at least $2k - 1$ wavelength interchangers.*

Proof. By contradiction, let C be a strictly nonblocking $k \times k$ WDM split cross-connect. As noted earlier, for such a cross-connect there exists a standard set of demands D_0 with routing R_0 . Let $M = 2k - 1$. If R_0 uses at least M wavelength interchangers, then we are done. Otherwise, by Lemma 6, we can repeatedly apply **Block** to produce a series of pairs $(D_{i+1}, R_{i+1}) = \mathbf{Block}(C, (D_i, R_i))$ where D_i is a standard set of demands and R_i is a routing of the demands in D_i as long as R_i has the property that it uses fewer than M wavelength interchangers. By Lemma 6 we know that $|W_{i+1}^F| = |W_i^F| - 1$ because the demand in Step 7 must be routed through a wavelength interchanger in W_i^F . Thus if $|W_0^F| = t$, then for some $i < t$, either R_i uses at least M wavelength interchangers and we are done or $|W_i^F| = 0$. If R_t uses fewer than M wavelength interchangers, then Lemma 6 guarantees that we can perform $\mathbf{Block}(C, (D_t, R_t))$. However, the demand in Step 7 must be serviced by a wavelength interchanger in W_t^F . Therefore Step 7 cannot be completed in $\mathbf{Block}(C, (D_t, R_t))$. Since C is strictly nonblocking it must be that R_t uses $2k - 1$ or more wavelength interchangers. \square

Theorem 7 says that any strictly nonblocking $k \times k$ WDM split cross-connect must have $2k - 1$ wavelength interchangers. Combining this with Theorem 4, we arrive at the following result for homogeneous cross-connects.

COROLLARY 8. *Any strictly nonblocking homogeneous $k \times k$ WDM cross-connect with $n > 1$ wavelengths must have at least $2k - 1$ wavelength interchangers.*

6.4. Other demand models for homogeneous cross-connects. In presenting the results of this section we have used a demand model in which the input and output wavelengths are specified as part of the demand. In this section we show that homogeneous cross-connects require $2k - 1$ wavelength interchangers even under less restrictive demand models. To prove this lower bound we must assume that $n \geq 4$. Although this is slightly weaker than our assumption in the previous sections that $n \geq 2$, it covers all cases that arise in practice.

For a fiber a and a routing R of existing demands, we define $f(a, R)$ to be the

set of wavelengths not in use on a by any route $r \in R$. We define a *general demand* $d = (I_x, O_y)$ to be a request for a connection from input fiber I_x to output fiber O_y . A *general route* r for d is a path from input fiber I_x to output fiber O_y such that the connection starts on an unused input wavelength, $\lambda_y \in f(I_x, R)$, stays on the same wavelength as long as it does not pass through a wavelength interchanger, and ends on unused output wavelength $\lambda_w \in f(O_y, R)$. We say that a cross-connect uses a *general demand model* if it supports general demands and general routes.

Note that considering a strictly nonblocking cross-connect in a general demand model actually weakens the definition of a strictly nonblocking cross-connect. Since a strictly nonblocking cross-connect must route any new demand regardless of the set of current routes in use and the wavelength assignment of those routes, one can view a demand model that specifies the input and output wavelengths as part of the demand as representing a worst-case adversary that requires not only a new connection from some input fiber to some output fiber but also chooses the *worst* wavelength assignment for the route chosen in the cross-connect. By allowing the cross-connect to choose the input and output wavelengths, the cross-connect is allowed to plan the choice of wavelengths. Therefore a cross-connect that supports only a general demand model is, to some extent, a hybrid between a strictly nonblocking cross-connect and a wide-sense nonblocking cross-connect. We consider such cross-connects and prove that by allowing the cross-connect, this added flexibility to choose the wavelength assignment of the routes does not change the number of wavelength interchangers needed in the cross-connect.

Note that although we allow the cross-connect to choose the input and output wavelengths, we still have the ability to force it to use a route that is known to satisfy a particular demand.

The outline for this section is similar to the earlier part of the section. We begin by reducing the problem to considering only split cross-connects. We then show that under certain conditions we can use a sequence of general demands to simulate a demand that specifies the input and output wavelengths as part of the demand. This allows us to prove that a strictly nonblocking $k \times k$ WDM split cross-connect supporting a general demand model requires $2k - 1$ wavelength interchangers and thus any strictly nonblocking $k \times k$ WDM cross-connect supporting a general demand model requires $2k - 1$ wavelength interchangers.

Thus, we begin this discussion showing that we can still reduce this problem to considering only split cross-connects. We assume throughout this section that $n \geq 4$. We define *LongGeneral* to be the set of strictly nonblocking WDM cross-connects using a general demand model that contain at least one directed path P from some input node $a \in I$ through $w_P > 1$ wavelength interchangers to some output node $b \in O$. We refer to P as a *long path* (see Figure 5). We define the operation **GeneralFill**(C, P, n) to route a set G of n demands from input fiber a to output fiber b along P .

LEMMA 9. *For any long path P with $w_P \leq n - 2$ and any routing chosen by the operation **GeneralFill**(C, P, n), there is a set S of demands such that*

1. $S \subset G$, and
2. S contains at most $n - 1$ demands,
3. *there is at least one wavelength λ_i such that for every section of fiber along P , there is some demand in S (possibly a different demand on each section) that is assigned λ_i along that section of fiber.*

Proof. By the assumptions of the lemma, P contains at most $n - 2$ wavelength interchangers. Therefore any route r along P can change wavelength at most $n - 2$

times. Therefore any particular demand can use at most $n - 1$ wavelengths. This implies that for any wavelength λ_i there must be at least one demand $d \in G$ that is not assigned λ_i anywhere along its route. Let $S = G \setminus \{d\}$. Since G contained n demands, each wavelength must be used by some demand on each section of P . Since d did not use λ_i on any section of P , the routes chosen for demands in S are such that for each section of fiber along P there is a route for some demand in S using λ_i . Therefore $S \subset G$ containing $n - 1$ demands and the routes for demands in S are such that for every section of fiber in P , there is some demand in S that has been assigned λ_i along that section of fiber. \square

Using Lemma 9, we can now prove that there does not exist a 2×2 WDM cross-connect with at most 2 wavelength interchangers such that those wavelength interchangers are on a long path. We will use this as a base case for a reduction that shows that we can always consider split cross-connects when showing that any $k \times k$ WDM cross-connect must have $2k - 1$ wavelength interchangers. Since a 1×1 WDM cross-connect cannot have both less than $2k - 1 = 1$ wavelength interchanger and a long path, we must use a 2×2 WDM cross-connect as our base case for the reduction.

THEOREM 10. *There does not exist a 2×2 WDM cross-connect $C \in LongGeneral$ with fewer than three wavelength interchangers.*

Proof. Assume that there exists a $C \in LongGeneral$ with two wavelength interchangers and $n \geq 4$. Let P be a long path from input fiber a to output fiber b with both wavelength interchangers in C on P . First, perform **GeneralFill**(C, P, n) to route a set G of n demands from a to b . Notice that after this has been done all wavelength interchangers along P have every wavelength in use on their input and output fibers. Since all the wavelength interchangers in C are on P , any new demands must not change wavelength. Let α be the other input fiber and β be the other output fiber. Suppose we now ask for n demands from α to β . Either the routing for these demands must use constant wavelength assignments for these demands or it contradicts the assumption that C has exactly two wavelength interchangers. Therefore assume that these new demands are routed with constant wavelength assignment.

By Lemma 9 there exists $S \subset G$ such that S contains at most $n - 1$ demands, such that for each section of fiber along P , there is some demand in S that has been assigned wavelength λ_i on that section. Remove the demand from α to β that is routed with constant wavelength assignment λ_i ; see Figure 12.

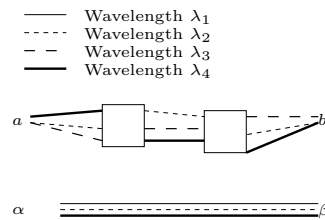


FIG. 12. A 2×2 WDM cross-connect C with $n = 4$ and a path P from input fiber a to output fiber b such that both wavelength interchangers in C are on P . There are currently three demands routed along P such that each section of P has at least one demand routed on wavelength λ_3 . There are three demands between α and β , none of which is assigned wavelength λ_3 on any fiber.

Furthermore, remove all demands along P that are not in S . At least one demand must be removed from P and that demand must have been routed such that it didn't use λ_i along any fiber in P . In particular it cannot have used λ_i on output fiber

b. Now add a demand from α to b . Since λ_i is the only wavelength available on α , any route for this demand must start with wavelength assignment λ_i . Since λ_i is not available on output fiber b this demand must change wavelength. However, each fiber along P and hence each input fiber to each wavelength interchanger along P already has a demand in S routed on it that is using wavelength λ_i . Therefore there is no wavelength interchanger that can be used to service this new demand and as a result either $C \notin \text{LongGeneral}$ or C must contain three or more wavelength interchangers which violates the assumption that C has at most two wavelength interchangers. \square

Recall that Theorem 4 considered the smallest k for which there was a $k \times k$ WDM cross-connect $C \in \text{Long}$ that had fewer than $2k - 1$ wavelength interchangers. It then used $\text{Fill}(C, P, n)$ to fill the path P with n demands. Once this path was filled with these demands, C was functionally equivalent to a $k' \times k'$ WDM cross-connect $C' \notin \text{Long}$ with fewer than $2k' - 1$ wavelength interchangers. Theorem 10 provides the base case for a similar reduction since it shows that there does not exist a cross-connect $C \in \text{LongGeneral}$ with $k = 2$ and at most two wavelength interchangers. Furthermore, the effect of $\text{GeneralFill}(C, P, n)$ when C is a cross-connect supporting a general demand model is to completely fill path P such that no new demands can use any of the fibers or any wavelength interchanger along P . Thus the following is a straightforward extension of Theorem 4.

COROLLARY 11. *If for some $k > 2$, there exists a $k \times k$ WDM cross-connect $C \in \text{LongGeneral}$ with $n \geq 4$ that has fewer than $2k - 1$ wavelength interchangers, then for some k' , where $1 < k' < k$, there exists a strictly nonblocking $k' \times k'$ WDM cross-connect $C' \notin \text{LongGeneral}$ that has fewer than $2k' - 1$ wavelength interchangers.*

Corollary 11 says that we need only consider whether there exists a strictly nonblocking $k \times k$ WDM *split* cross-connect supporting a general demand model that uses fewer than $2k - 1$ wavelength interchangers. Therefore recall the version of $\text{Block}(C, (D_i, R_i))$ that requires that demands specify the input and output wavelengths. We assume that R_i uses at most $2k - 2$ wavelength interchangers and every demand in D_i has input wavelength λ_1 or λ_2 and output wavelength λ_1 or λ_2 .

$\text{Block}(C, (D_i, R_i))$.

1. Take two wavelength interchangers, WI_u and WI_v , in W_i^B that, by Lemma 5, each service two demands.
2. $\text{Uncross}(WI_u)$ and $\text{Uncross}(WI_v)$.
3. Let $(I_{u1}, \lambda_1, O_{u1}, \lambda_1)$ and $(I_{u2}, \lambda_2, O_{u2}, \lambda_2)$ be the two resulting demands that WI_u services. Let $(I_{v1}, \lambda_1, O_{v1}, \lambda_1)$ and $(I_{v2}, \lambda_2, O_{v2}, \lambda_2)$ be the two resulting demands that WI_v services.
4. Remove $(I_{u1}, \lambda_1, O_{u1}, \lambda_1)$ and $(I_{v2}, \lambda_2, O_{v2}, \lambda_2)$ from D_i and route all remaining demands according to R_i to create D_{i+1} and R_{i+1} .
5. Add $(I_{v2}, \lambda_2, O_{u1}, \lambda_1)$ to D_{i+1} and add a valid route for this demand to R_{i+1} .
6. Add $(I_{u1}, \lambda_1, O_{v2}, \lambda_2)$ to D_{i+1} and add a valid route for this demand to R_{i+1} .
7. Return (D_{i+1}, R_{i+1}) .

Consider Step 5 which asks for a route from I_{v2} to O_{u1} such that the demand is routed on input wavelength λ_2 on input fiber I_{v2} and output wavelength λ_1 on output fiber O_{u1} . In the current model we cannot specify the input or output wavelength. Therefore we need to replace Step 5 with a sequence of general demands $\{d_1, d_2, \dots, d_j\}$ such that the final general demand, d_j , is from input fiber I_{v2} to output fiber O_{u1} and the routes for the demands $\{d_1, d_2, \dots, d_{j-1}\}$ require that demand d_j uses input wavelength λ_2 and output wavelength λ_1 . Once d_j has been routed through C , we can remove demands $\{d_1, d_2, \dots, d_{j-1}\}$ and be left with the same re-

sult as Step 5. In order to show that this is possible, we will use the fact that the set of demands D_i and routing R_i of D_i are such that all demands are routed using either input wavelength λ_1 or λ_2 and either output wavelength λ_1 or λ_2 .

More generally, for a set of demands, D , we will define a routing R of D to be **Constrained** (λ_i, λ_j) if for every demand $d \in D$, the route r for d uses either λ_i or λ_j as the input wavelength and either λ_i or λ_j as the output wavelength. Notice that the advantage of maintaining a **Constrained** (λ_i, λ_j) routing is that we are guaranteed that all other wavelengths are unused on all the fibers. Since $n \geq 4$ this implies that at most half of the wavelengths are in use on any given input or output fiber under any **Constrained** (λ_i, λ_j) routing.

Consider a cross-connect C and suppose there is a set of demands, D , where the routing R of D is **Constrained** (λ_i, λ_j) . Furthermore, assume that there is an input fiber I_a with $\lambda_i \in f(I_a, R)$ and there is an output fiber O_b with $\lambda_j \in f(O_b, R)$. We create the following operation **Force** $(C, (D, R), (I_a, \lambda_i, O_b, \lambda_j))$ that forces the route for a new general demand from I_a to O_b to use input wavelength λ_i and output wavelength λ_j . In describing **Force** $(C, (D, R), (I_a, \lambda_i, O_b, \lambda_j))$ we assume that $|f(I_a, R)| \geq |f(O_b, R)|$. The case when $|f(I_a, R)| < |f(O_b, R)|$ is analogous.

Force $(C, (D, R), (I_a, \lambda_i, O_b, \lambda_j))$.

1. Assume $|f(I_a, R)| \geq |f(O_b, R)|$.
2. Create $|f(O_b, R)|$ demands from input fiber I_a to output fiber O_b .
3. If a demand from Step 2 is routed on output wavelength λ_j , then remove it.
4. Let D' and R' be the current set of demands and current routing of these demands.
5. Create $|f(I_a, R')|$ demands from input fiber I_a to output fibers other than O_b .
6. Remove the demand that uses input wavelength λ_i on input fiber I_a and redefine D' and R' to be the current set of demands and routing of these demands.
7. Add $|f(O_b, R')|$ demands from input fibers other than I_a to output fiber O_b .
8. Remove the demand from output fiber O_b that is routed on output wavelength λ_j .
9. Add a demand from I_a to O_b to D' and add the route r for this demand to R' .
10. Remove any existing demands that were made in Steps 2, 5, and 7.
11. Return the current set of demands, D' , and the current routing R' of D' .

LEMMA 12. *Let C be a homogeneous strictly nonblocking $k \times k$ WDM split cross-connect with $n \geq 4$ available wavelengths. Let D be a set of demands and R a set of routes that are **Constrained** (λ_i, λ_j) . If $\lambda_i \in f(I_a, R)$ and $\lambda_j \in f(O_b, R)$, then*

1. $(D', R') = \mathbf{Force}(C, (D, R), (I_a, \lambda_i, O_b, \lambda_j))$ can be performed,
2. $D' = \{d\} \cup D$, where d is a demand from input fiber I_a to output fiber O_b ,
3. $R' = \{r\} \cup R$, where r is the route for d using input wavelength λ_i and output wavelength λ_j , and
4. R' is **Constrained** (λ_i, λ_j) .

Proof. Notice that the case when $|f(I_a, R)| < |f(O_b, R)|$ is analogous to the case when $|f(I_a, R)| > |f(O_b, R)|$. Therefore without loss of generality we consider $|f(I_a, R)| \geq |f(O_b, R)|$. Note that since R was **Constrained** (λ_i, λ_j) , each fiber can have at most two demands and therefore $n - 2 \leq |f(I_a, R)| \leq n$ and $n - 2 \leq |f(O_b, R)| \leq n$.

In Step 2 we make $|f(O_b, R)|$ demands. If $|f(I_a, R)| = |f(O_b, R)|$, then all free input wavelengths will be used on I_a . Otherwise, if $|f(I_a, R)| > |f(O_b, R)|$, then

one wavelength on input fiber I_a will be unused. We then remove the demand that used output wavelength λ_j . Therefore, before Step 5, $|f(I_a, R')| \leq 2$. Since R was **Constrained** (λ_i, λ_j) and since we have added only demands from output fiber O_b , all other output fibers must still have at least two wavelengths unused prior to Step 5. Therefore there are enough output wavelengths to make the requested demands in Step 5. After Step 6 there will be only one free wavelength, λ_i , on input fiber I_a and at most two free wavelengths on output fiber O_b . Therefore, by an analogous argument, there are enough free input wavelengths to make the demands requested in Step 7. Furthermore, prior to Step 7 we removed any demand from output fiber O_b that was routed on output wavelength λ_j . Therefore the demand we remove in Step 8 that is routed on output wavelength λ_j on output fiber O_b is guaranteed to use an input fiber other than I_a . Hence after Step 8, input fiber I_a has only input wavelength λ_i available and output fiber O_b has only output wavelength λ_j available. Therefore the route for the demand made in Step 9 will use input wavelength λ_i and output wavelength λ_j . Finally, since we remove all demands added in Steps 2, 5, and 7, $D' = \{d\} \cup D$, where d is a demand from input fiber I_a to output fiber O_b and R' is **Constrained** (λ_i, λ_j) . \square

Notice that Lemma 12 allows us to create a set of standard demands, D , and a set of routes R for those demands. Furthermore, Lemma 12 allows us to replace the demands made in Steps 2, 5, and 6 of $\text{Block}(C, (D_i, R_i))$ with the appropriate sequence of general demands. Therefore we arrive at the following corollary to Theorem 7.

COROLLARY 13. *For any homogeneous strictly nonblocking $k \times k$ WDM split cross-connect using a general demand model with $k \geq 2$ and $n \geq 4$, there must be a standard set of demands, D_t , and a standard routing R_t of D_t that uses at least $2k - 1$ wavelength interchangers.*

Combining the results in Corollaries 11 and 13 implies the following result for homogeneous cross-connects.

COROLLARY 14. *Any homogeneous strictly nonblocking $k \times k$ WDM cross-connect with $k \geq 2$ and $n \geq 4$ wavelengths supporting any demand model must have at least $2k - 1$ wavelength interchangers.*

7. Determining the optimal number of wavelength interchangers for heterogeneous split cross-connects. In this section we focus on heterogeneous split cross-connects. Recall that a heterogeneous $k_1 \times k_2$ WDM cross-connect has k_1 input fibers that each support n_1 input wavelengths and k_2 output fibers that each support n_2 output wavelengths. Note that the set of input and output wavelengths can be completely disjoint. Therefore we now refer to the output wavelengths as $\{\gamma_1, \gamma_2, \dots, \gamma_{n_2}\}$. Without loss of generality, we assume that $k_1 \leq k_2$.

In section 5 we presented a family of strictly nonblocking $k_1 \times k_2$ WDM cross-connects that use $\min(k_1 + k_2 - 1, n_1 k_1)$ wavelength interchangers. In section 6 we showed that this is the optimal number of wavelength interchangers for any $k \times k$ WDM homogeneous cross-connect. In this section we extend the techniques of section 6 to $k_1 \times k_2$ WDM heterogeneous split cross-connects. To show that any heterogeneous $k_1 \times k_2$ WDM split cross-connect requires $\min(k_1 + k_2 - 1, n_1 k_1)$ wavelength interchangers we consider any strictly nonblocking $k_1 \times k_2$ WDM split cross-connect with $m < \min(k_1 + k_2 - 1, n_1 k_1)$ wavelength interchangers and show that there is a sequence of demands and a routing of those demands such that there is a final valid demand d that cannot be routed by the cross-connect. For the heterogeneous case we consider only split cross-connects and only a demand model where the input and output wavelengths are specified as part of the demand.

The proof that any heterogeneous strictly nonblocking $k_1 \times k_2$ WDM split cross-connect must have $\min(k_1 + k_2 - 1, n_1 k_1)$ wavelength interchangers has a flavor similar to the proof, presented in the previous section, that showed that any homogeneous $k \times k$ WDM cross-connect requires $2k - 1$ wavelength interchangers. We present the lower bound in two cases. First, we consider $k_1 \times k_2$ WDM cross-connects where $n_1 k_1 \leq k_2$. In this case we can create a set of demands that all use the same output wavelength and therefore must all use their own wavelength interchanger. Hence we arrive at a simple lower bound that says that any strictly nonblocking $k_1 \times k_2$ WDM cross-connect must have at least $n_1 k_1$ wavelength interchangers if $n_1 k_1 \leq k_2$. We then consider cross-connects where $n_1 k_1 > k_2$. Our basic idea is to mimic the proof from section 6 that showed that any $k \times k$ WDM cross-connect requires $2k - 1$ wavelength interchangers. To achieve this for heterogeneous cross-connects we use two input wavelengths, $\{\lambda_1, \lambda_2\}$, two output wavelengths $\{\gamma_1, \gamma_2\}$, and a set of input and output fibers of equal size, say z , to force $2z - 1$ wavelength interchangers to service either a demand with input wavelength λ_1 or a demand with output wavelength γ_2 . We use the remaining $k_2 - z$ output fibers and the unused wavelengths on the input fibers to “hold” $k_2 - z$ wavelength interchangers. By making z as large as possible while still having enough available input wavelengths to make the $k_2 - z$ demands, we are able to show that any strictly nonblocking $k_1 \times k_2$ WDM split cross-connect requires $\min(k_1 + k_2 - 1, n_1 k_1)$ wavelength interchangers.

7.1. Lower bound for heterogeneous split cross-connects. We approach the lower bound for the number of wavelength interchangers in a strictly nonblocking $k_1 \times k_2$ WDM cross-connect as follows.

1. We show that if $k_2 \geq n_1 k_1$, then $n_1 k_1$ wavelength interchangers are necessary.
2. We show that when $k_2 < n_1 k_1$, any strictly nonblocking $k_1 \times k_2$ WDM cross-connect must have at least $\min(k_1 + k_2 - 1, n_1 k_1 - 1)$ wavelength interchangers.
3. We then consider the case when $(n_1 - 1)k_1 < k_2 < n_1 k_1$, where the previous step only proved that $n_1 k_1 - 1$ wavelength interchangers are necessary, and show that in fact $n_1 k_1$ wavelength interchangers are necessary for any strictly nonblocking $k_1 \times k_2$ WDM cross-connect.

We begin this section by considering the case when $k_2 \geq n_1 k_1$ where we saw in section 5 that there exists a strictly nonblocking WDM cross-connect that uses $n_1 k_1$ wavelength interchangers.

THEOREM 15. *If C is a strictly nonblocking $k_1 \times k_2$ WDM cross-connect with $k_2 \geq n_1 k_1$, then C must have at least $n_1 k_1$ wavelength interchangers.*

Proof. If $k_2 \geq n_1 k_1$, then a valid demand set could include $n_1 k_1$ demands that all use the same output wavelength. Since no wavelength interchanger can service more than one of these demands, any $k_1 \times k_2$ WDM cross-connect will need at least $n_1 k_1$ wavelength interchangers if it is strictly nonblocking. \square

If $k_2 < n_1 k_1$, it is easy to see by the argument in the proof of Theorem 15 that any strictly nonblocking WDM cross-connect must have at least k_2 wavelength interchangers. In order to show that any strictly nonblocking WDM split cross-connect actually requires $\min(k_1 + k_2 - 1, n_1 k_1)$ wavelength interchangers, we first show that $\min(k_1, n_1 k_1 - k_2) + k_2 - 1 = \min(k_1 + k_2 - 1, n_1 k_1 - 1)$ wavelength interchangers are needed. To do this we consider any strictly nonblocking WDM split cross-connect C with fewer than $\min(k_1, n_1 k_1 - k_2) + k_2 - 1$ wavelength interchangers. For such a cross-connect we will create a demand set D and a routing R of D such that C must use each wavelength interchanger to service a demand with either input wavelength λ_1 or output wavelength γ_2 . Given D and R we will then show that one additional

valid demand $d \notin D$ exists with input wavelength λ_1 and output wavelength γ_2 . By construction, all the wavelength interchangers in the cross-connect will already be servicing a demand that either uses this new demand's input wavelength or its output wavelength. Therefore the cross-connect will not have a wavelength interchanger to service the new demand. This implies that if a $k_1 \times k_2$ WDM split cross-connect is strictly nonblocking, then it must have at least $\min(k_1 + k_2 - 1, n_1 k_1 - 1)$ wavelength interchangers. For cases where this construction shows that $n_1 k_1 - 1$ wavelength interchangers are necessary, we then start with the routing that uses $n_1 k_1 - 1$ wavelength interchangers and then manipulate the demands to require $n_1 k_1$ wavelength interchangers for any $k_1 \times k_2$ WDM cross-connect.

Consider a strictly nonblocking $k_1 \times k_2$ WDM cross-connect C where $k_2 < n_1 k_1$. Let $z = \min(k_1, n_1 k_1 - k_2)$, $\mathcal{I}^A = \{a_1, a_2, \dots, a_z\}$ be a subset of I consisting of z input fibers, $\mathcal{O}^A = \{b_1, b_2, \dots, b_z\}$ be a subset of z output fibers in O , and $\mathcal{O}^H = \{h_1, h_2, \dots, h_{k_2-z}\}$ be the remaining output fibers. Given this partition of the input and output fibers we say that a valid demand set D is *standard* if and only if

1. $|D| = k_2 + z$,
2. $2z$ of the demands in D are demands with an input fiber in \mathcal{I}^A , an output fiber in \mathcal{O}^A , input wavelength either λ_1 or λ_2 and output wavelength either γ_1 or γ_2 , and
3. the other $k_2 - z$ demands in D have output wavelength γ_2 and an output fiber in \mathcal{O}^H .

See Figure 13 for an example of a standard set of demands and a standard routing of those demands on a heterogeneous cross-connect C .

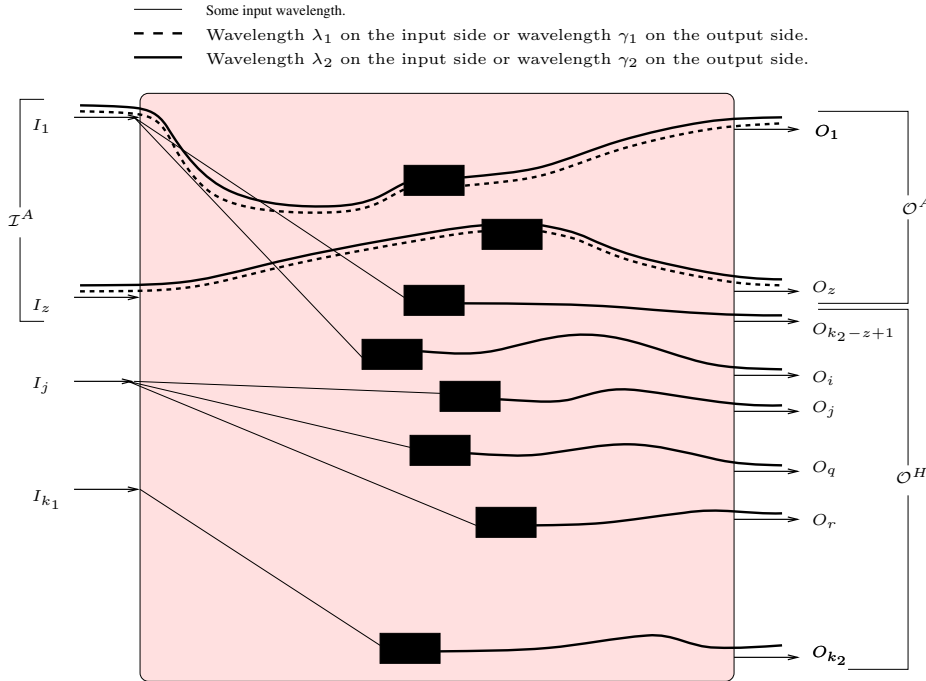


FIG. 13. A set of standard demands, D_0 , on C .

We will be constructing various standard demand sets D_i in what follows. For such a standard set of demands, D_i , define D_i^A to be the subset of demands in D_i

with an output fiber in \mathcal{O}^A . The subset of demands in D_i with an output fiber in \mathcal{O}^H will remain fixed for all i and we denote this set by D^H .

As with our construction for homogeneous cross-connects, we will assume that we have a routing R_i for a standard set of demands, D_i . We will then successively transform the standard demand sets into new standard demand sets by creating new demands with input wavelength λ_1 and output wavelength γ_2 with the goal of forcing these new demands to use wavelength interchangers that are so far unaccounted for. To do this for heterogeneous cross-connects we must partition the set of wavelength interchangers in C into three sets depending on the types of demands that they service. The routes for the fixed demands in D^H will be unchanging and the fixed set of wavelength interchangers that service these routes is denoted by W^H . The intuition is that these wavelength interchangers will always be “held” by these routes and so the cross-connect will be unable to route other demands with output wavelength γ_2 through them. We modify the definition of W_i^B from the homogeneous case to be the set of wavelength interchangers that *are not in* W^H yet service a demand with either input wavelength λ_1 and/or output wavelength γ_2 . These wavelength interchangers are thought of as “blocking” demands with input wavelength λ_1 and output wavelength γ_2 from being routed through them. Finally, let W_i^F be the set of all other wavelength interchangers. This set of wavelength interchangers is thought to be “free” to service demands with input wavelength λ_1 and output wavelength γ_2 .

We define a valid routing R_i of a standard set of demands D_i to be *standard* if

1. each wavelength interchanger in W^H services only one demand and
2. the set of output paths of the routes for demands in D^H are edge disjoint from all output paths of the routes for demands in D_i^A .

Suppose C is a strictly nonblocking $k_1 \times k_2$ WDM split cross-connect where $k_2 < n_1 k_1$. Then we show that a standard set of demands, D_0 , and a standard routing R_0 for D_0 exist within C . First, create $z = \min(k_1, n_1 k_1 - k_2)$ demands of the form $d_{i1} = (a_i, \lambda_1, b_i, \gamma_1)$ for $1 \leq i \leq z$. Since C is strictly nonblocking, C must be able to satisfy these z demands. Notice also that these demands must be routed on edge disjoint paths because they all have the same input and output wavelengths. For $1 \leq i \leq z$, create a demand $d_{i2} = (a_i, \lambda_2, b_i, \gamma_2)$ routed along the same path as d_{i1} .

Now with these $2z$ demands on the cross-connect, we would like to create $k_2 - z$ demands such that each output fiber $h_j \in \mathcal{O}^H$ has a demand with output wavelength γ_2 . First, consider the case where $k_1 \leq n_1 k_1 - k_2$ and hence $z = k_1$. In this case $(n_1 - 1)k_1 \geq k_2$. Thus, $(n_1 - 2)k_1 \geq k_2 - k_1$. Since there are $n_1 k_1 - 2k_1$ input wavelengths unused on the set of input fibers, clearly there are enough input wavelengths available to make $k_2 - k_1$ new valid demands all using output wavelength γ_2 . On the other hand, if $z = n_1 k_1 - k_2$, then $n_1 k_1 - 2z = k_2 - z$, and therefore there are again enough available wavelengths on the input side to make $k_2 - z$ demands with output wavelength γ_2 . Therefore we add $k_2 - z$ valid demands such that each fiber $h_j \in \mathcal{O}^H$ has a demand with output wavelength γ_2 .

Clearly the set of demands that use output wavelength γ_2 must all have edge disjoint output paths. Since all other demands in D_0^A are routed along a path that also has a demand in D_0^A with output wavelength γ_2 , all demands in D_0^A are routed along output paths that are edge disjoint with respect to all output paths of routes for demands in D^H . Therefore D_0 is a standard set of demands and the routing described is a standard routing R_0 for D_0 .

LEMMA 16. *If C is a strictly nonblocking $k_1 \times k_2$ WDM split cross-connect, D_i is a standard set of demands, and R_i is a standard routing of the demands in D_i that uses $\min(k_1, n_1 k_1 - k_2) + k_2 - g$ wavelength interchangers, where $g > 0$, then g wavelength interchangers in W_i^B will each service two demands both of whose input wavelengths are λ_1 and λ_2 and whose output wavelengths are γ_1 and γ_2 .*

Proof. By definition, any standard routing R_i of D_i routes at most one demand through any wavelength interchanger in W^H and therefore that demand must be a demand whose output fiber is in \mathcal{O}^H . There are $k_2 - z$ of these demands and therefore $|W^H| = k_2 - z$. Since the total number of wavelength interchangers used is $\min(k_1, n_1 k_1 - k_2) + k_2 - g = z + k_2 - g$ and $|W^H| = k_2 - z$, it must be the case that the number of wavelength interchangers in W_i^B and W_i^F is $2z - g$. The $2z$ demands of D_i^A must be routed through wavelength interchangers in W_i^B or W_i^F since R_i is a standard routing. These $2z$ demands in D_i^A use only input wavelengths λ_1 and λ_2 and output wavelengths γ_1 and γ_2 . Thus no wavelength interchanger can service more than two of these demands. This implies that g wavelength interchangers in W_i^B or W_i^F must service two demands. Any such wavelength interchanger that services two of these demands must service a demand with input wavelength λ_1 and a demand with output wavelength γ_2 and therefore is by definition in W_i^B . \square

Given the standard set of demands, D_0 , and the routing R_0 of D_0 , we now present a manipulation, similar to $\text{Block}(C, (D_i, R_i))$, that can be used to iteratively change the set of demands on C so that eventually we arrive at a standard set of demands, D_t , and a standard routing R_t of D_t such that every wavelength interchanger in C is servicing a demand with either input wavelength λ_1 or output wavelength γ_2 . Notice that this is equivalent to showing that if C has no more than $\min(k_1, n_1 k_1 - k_2) + k_2 - g$ wavelength interchangers for some $g > 0$, then a standard set of demands D_i and a standard routing R_i of D_i exist such that $|W^H| = k_2 - z$, $|W_i^B| = 2z - g$ and $|W_i^F| = 0$.

Let $WI_j \in W_i^B$ be a wavelength interchanger that services exactly two demands, d_1 and d_2 . Suppose these demands have the form $d_1 = (a_1, \lambda_1, b_2, \gamma_2)$ and $d_2 = (a_2, \lambda_2, b_1, \gamma_1)$. Recall the operation $\text{Uncross}(WI_j)$ which is defined to have the effect of changing these two demands to be $(a_1, \lambda_1, b_1, \gamma_1)$ and $(a_2, \lambda_2, b_2, \gamma_2)$ and routing these new demands exactly as d_1 and d_2 were routed while keeping all other demands and routes unchanged. Again, notice that the only real change made by $\text{Uncross}(WI_j)$ is to swap the output paths of the routes of d_1 and d_2 . On the other hand, if the demands have the form $d_1 = (a_1, \lambda_1, b_2, \gamma_1)$ and $d_2 = (a_2, \lambda_2, b_1, \gamma_2)$, then $\text{Uncross}(WI_j)$ has no effect; see Figure 10.

We define the operation $\text{HeterogeneousBlock}(C, (D_i, R_i))$, which is similar to $\text{Block}(C, (D_i, R_i))$. However, now we assume that C is a heterogeneous strictly nonblocking $k_1 \times k_2$ WDM split cross-connect, D_i is a standard set of demands, and R_i is a standard routing of the demands in D_i that uses fewer than $\min(k_1, n_1 k_1 - k_2) + k_2 - 1$ wavelength interchangers. Each time we run $\text{HeterogeneousBlock}(C, (D_i, R_i))$ we alter D_i and R_i to create a new standard set of demands D_{i+1} and a standard routing R_{i+1} for D_{i+1} such that the number of wavelength interchangers that service a demand with either input wavelength λ_1 or output wavelength γ_2 is strictly greater under D_{i+1} and R_{i+1} than it was under D_i and R_i . In section 6 we saw the operation $\text{Block}(C', (D_i, R_i))$ which performed a similar task for a homogeneous cross-connect C' . In $\text{Block}(C', (D_i, R_i))$ we did not have “extra” demands, D^H , that “held” the wavelength interchangers in W^H . For a heterogeneous cross-connect we use these demands to block the wavelength interchangers in W^H from servicing the two new demands that we create each iteration. In order to prevent either demand from being

routed through one of the wavelength interchangers in W^H , we switch the output wavelength of the demands in D^H such that the demands in D^H always use the same output wavelength as the new demand that we are creating; see Figure 14.

After we define **HeterogeneousBlock**($C, (D_i, R_i)$), we will show that it consists of a sequence of valid operations.

HeterogeneousBlock($C, (D_i, R_i)$).

1. Take two wavelength interchangers, WI_u and WI_v , in W_i^B that, by Lemma 16, each service two demands.
2. **Uncross**(WI_u) and **Uncross**(WI_v).
3. Let $(a_{u1}, \lambda_1, b_{u1}, \gamma_1)$ and $(a_{u2}, \lambda_2, b_{u2}, \gamma_2)$ be the two resulting demands that WI_u services. Let $(a_{v1}, \lambda_1, b_{v1}, \gamma_1)$ and $(a_{v2}, \lambda_2, b_{v2}, \gamma_2)$ be the two resulting demands that WI_v services.
4. Remove $(a_{u1}, \lambda_1, b_{u1}, \gamma_1)$ and $(a_{v2}, \lambda_2, b_{v2}, \gamma_2)$ from D_i and route all remaining demands according to R_i to create D_{i+1} and R_{i+1} .
5. For any demand $d_j \in D_{i+1}$ of the form $(I_e, \lambda_t, h_j, \gamma_2)$, remove d_j from D_{i+1} , replace it with the demand $d'_j = (I_e, \lambda_t, h_j, \gamma_1)$, and route d'_j along the same path that d_j was routed along.
6. Add $(a_{v2}, \lambda_2, b_{u1}, \gamma_1)$ to D_{i+1} and add a valid route for this demand to R_{i+1} .
7. For any demand $d'_j \in D_{i+1}$ of the form $(I_e, \lambda_t, h_j, \gamma_1)$, remove d'_j from D_{i+1} , replace it with demand $d_j = (I_e, \lambda_t, h_j, \gamma_2)$, and route d_j along the same path that d'_j was routed along.
8. Add $(a_{u1}, \lambda_1, b_{v2}, \gamma_2)$ to D_{i+1} and add a valid route for this demand to R_{i+1} .
9. Return (D_{i+1}, R_{i+1}) .

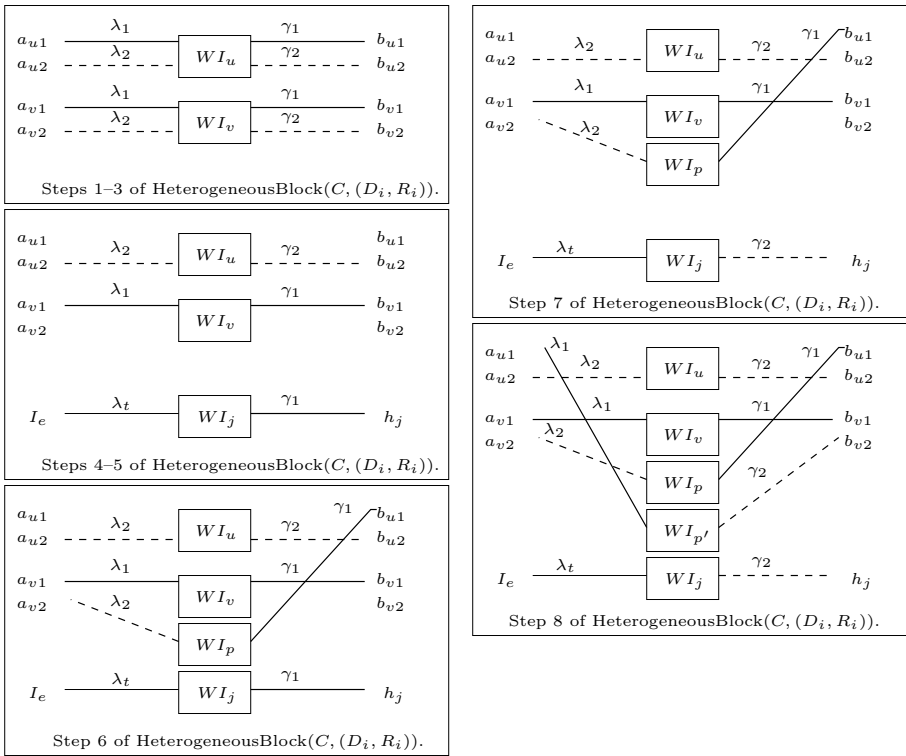


FIG. 14. The steps in *HeterogeneousBlock*($C, (D_i, R_i)$).

The initial set of demands on C is D_0 and these are routed according to R_0 as defined above. We inductively define $(D_{i+1}, R_{i+1}) = \mathbf{HeterogeneousBlock}(C, (D_i, R_i))$ as long as R_i uses fewer than $\min(k_1, n_1 k_1 - k_2) + k_2 - 1$ wavelength interchangers. Consider $\mathbf{HeterogeneousBlock}(C, (D_i, R_i))$. We now prove that each step can be performed. Consider Steps 1–3.

1. Take two wavelength interchangers, WI_u and WI_v , in W_i^B that, by Lemma 16, each service two demands.
2. $\mathbf{Uncross}(WI_u)$ and $\mathbf{Uncross}(WI_v)$.
3. Let $(a_{u1}, \lambda_1, b_{u1}, \gamma_1)$ and $(a_{u2}, \lambda_2, b_{u2}, \gamma_2)$ be the two resulting demands that WI_u services. Let $(a_{v1}, \lambda_1, b_{v1}, \gamma_1)$ and $(a_{v2}, \lambda_2, b_{v2}, \gamma_2)$ be the two resulting demands that WI_v services.

By Lemma 16 and the assumption that R_i uses fewer than $\min(k_1, n_1 k_1 - k_2) + k_2 - 1$ wavelength interchangers, there must exist two wavelength interchangers in W_i^B that service two demands. Given these two wavelength interchangers, we then “uncross” their demands. This does not change any of the wavelengths used along any of the fibers in C . In particular, while this alters the set of demands, the new set of demands is still a standard set of demands.

In Step 4, we remove two demands and leave the remaining demands unchanged. By our choice of demands to remove, we ensure that WI_v and WI_u will still each service one demand with either input wavelength λ_1 or output wavelength γ_2 . Thus WI_v and WI_u remain in W_i^B .

4. Remove $(a_{u1}, \lambda_1, b_{u1}, \gamma_1)$ and $(a_{v2}, \lambda_2, b_{v2}, \gamma_2)$ from D_i and route all remaining demands according to R_i to create D_{i+1} and R_{i+1} .

Next in Step 5, we change the output wavelength of every demand in D^H to be γ_1 . Since R_i is a standard routing, all the output paths for demands in D^H are edge disjoint from all other demands. Therefore this step does not create an invalid set of demands or routings for those demands. Changing the output wavelength of every demand in D^H means that each wavelength interchanger in W^H is servicing a demand with output γ_1 when a route for the new demand with output wavelength γ_1 in Step 6 is found.

5. For any demand $d_j \in D_{i+1}$ of the form $(I_e, \lambda_t, h_j, \gamma_2)$, remove d_j from D_{i+1} , replace it with the demand $d'_j = (I_e, \lambda_t, h_j, \gamma_1)$, and route d'_j along the same path that d_j was routed along.
6. Add $(a_{v2}, \lambda_2, b_{u1}, \gamma_1)$ to D_{i+1} and add a valid route for this demand to R_{i+1} .

As a result this demand can be serviced only by a wavelength interchanger in either W_i^B or W_i^F and any route for this demand will use an output path that is edge disjoint from the output paths for all demands in D^H . If the wavelength interchanger that does service the demand from Step 6 is in W_i^F , then it will be in W_{i+1}^F , and clearly if it is in W_i^B , then it will be in W_{i+1}^B . So after Step 6, $|W_{i+1}^B| = |W_i^B|$, $|W_{i+1}^F| = |W_i^F|$ and $|W^H|$ of course remains fixed.

Next, in Step 7, we switch all $d'_j = (I_e, \lambda_t, h_j, \gamma_1)$ back to $d_j = (I_e, \lambda_t, h_j, \gamma_2)$. Again this is possible since R_i is a standard routing and the output paths of the routes for these demands must be edge disjoint from all other output paths. Furthermore, switching the output wavelength of each of the demands serviced by a wavelength interchanger in W^H back to γ_2 means that the new demand in Step 8 must be serviced by a wavelength interchanger in either W_i^B or W_i^F and the route chosen to service the demand will have an edge disjoint output path from all output paths for demands in D^H .

7. For any demand $d'_j \in D_{i+1}$ of the form $(I_e, \lambda_t, h_j, \gamma_1)$, remove d'_j from D_{i+1} ,

replace it with demand $d_j = (I_e, \lambda_t, h_j, \gamma_2)$, and route d_j along the same path that d'_j was routed along.

8. Add $(a_{u1}, \lambda_1, b_{v2}, \gamma_2)$ to D_{i+1} and add a valid route for this demand to R_{i+1} .

A valid route for the new demand in Step 8 must exist since C is strictly non-blocking. Furthermore, a wavelength interchanger in W_i^F must be used to service this demand since all wavelength interchangers in W_i^B are servicing a demand with either input wavelength λ_1 or output wavelength γ_2 , and all wavelength interchangers in W^H are servicing a demand with output wavelength γ_2 . Since the demand requested in Step 8 has both input wavelength λ_1 and output wavelength γ_2 , the wavelength interchanger that services this new demand will be in W_{i+1}^B . Thus, after Step 8, D_{i+1} is a standard set of demands and R_{i+1} is a standard routing for those demands such that $|W^H|$ remains fixed, $|W_{i+1}^B| = |W_i^B| + 1$, and $|W_{i+1}^F| = |W_i^F| - 1$. Hence we conclude the following lemma.

LEMMA 17. *If C is a strictly nonblocking $k_1 \times k_2$ WDM split cross-connect with $k_2 < n_1 k_1$, D_i is a standard set of demands, and R_i is a standard routing of the demands in D_i that uses fewer than $\min(k_1, n_1 k_1 - k_2) + k_2 - 1$ wavelength interchangers, then*

1. **HeterogeneousBlock** $(C, (D_i, R_i))$ can be executed,
2. $(D_{i+1}, R_{i+1}) = \mathbf{HeterogeneousBlock}(C, (D_i, R_i))$,
3. D_{i+1} is a standard set of demands,
4. R_{i+1} is a standard routing of the demands in D_{i+1} ,
5. $|W^H|$ remains fixed,
6. $|W_{i+1}^B| = |W_i^B| + 1$, and
7. $|W_{i+1}^F| = |W_i^F| - 1$.

We now apply Lemma 17 to show that any strictly nonblocking $k_1 \times k_2$ WDM split cross-connect with $k_1 \leq k_2 < n_1 k_1$ must have at least $\min(k_1, n_1 k_1 - k_2) + k_2 - 1$ wavelength interchangers.

THEOREM 18. *For any strictly nonblocking $k_1 \times k_2$ WDM split cross-connect with $k_1 \leq k_2 < n_1 k_1$, there must be a standard set of demands D_t and a standard routing R_t of D_t that uses at least $\min(k_1, n_1 k_1 - k_2) + k_2 - 1 = \min(k_1 + k_2 - 1, n_1 k_1 - 1)$ wavelength interchangers.*

Proof. By contradiction, let C be a strictly nonblocking $k_1 \times k_2$ WDM split cross-connect where $k_1 \leq k_2 < n_1 k_1$. Start with the standard set of demands D_0 with standard routing R_0 , which must exist. Let $M = \min(k_1, n_1 k_1 - k_2) + k_2 - 1$. If R_0 uses at least M wavelength interchangers, then we are done. Otherwise, by Lemma 17, we can repeatedly apply **HeterogeneousBlock** to produce a series of pairs $(D_{i+1}, R_{i+1}) = \mathbf{HeterogeneousBlock}(C, (D_i, R_i))$ where each D_i is a standard set of demands and each R_i is a standard routing of the demands in D_i . This sequence continues as long as R_i has the property that it uses fewer than M wavelength interchangers. By Lemma 17 we know that $|W_{i+1}^F| = |W_i^F| - 1$ because the demand in Step 9 must be routed through a wavelength interchanger in W_i^F . Thus if $|W_0^F| = t$, then either for some $i < t$, R_i uses at least M wavelength interchangers and we are done or $|W_t^F| = 0$. If R_t uses fewer than M wavelength interchangers, then Lemma 17 guarantees that we can perform **HeterogeneousBlock** $(C, (D_t, R_t))$. However, the demand in Step 9 must be serviced by a wavelength interchanger in W_t^F . Therefore Step 9 cannot be completed in **HeterogeneousBlock** $(C, (D_t, R_t))$. Since C is strictly nonblocking it must be that R_t uses $\min(k_1, n_1 k_1 - k_2) + k_2 - 1$ or more wavelength interchangers. \square

Theorem 18 says that if $k_2 \leq (n_1 - 1)k_1$, then $k_1 + k_2 - 1$ wavelength interchangers are needed. However, if $(n_1 - 1)k_1 < k_2 < n_1k_1$, then it says only that $n_1k_1 - 1$ wavelength interchangers are needed. We now show that in this case, there must, in fact, be at least n_1k_1 wavelength interchangers.

We could easily show that n_1k_1 wavelength interchangers are necessary, if we could continue executing $(D_{i+1}, R_{i+1}) = \mathbf{HeterogeneousBlock}(C, (D_i, R_i))$ until R_{i+1} uses $n_1k_1 - 1$ wavelength interchangers and $W_{i+1}^F = 0$. However, in general, it is possible that at some point $\mathbf{HeterogeneousBlock}(C, (D_i, R_i))$ could return a standard set of demands D_{i+1} and a standard routing R_{i+1} for D_{i+1} such that $n_1k_1 - 1$ wavelength interchangers in C are used to service the demands but $W_{i+1}^F > 0$. Once this many wavelength interchangers in C are used to service a demand we can no longer run $\mathbf{HeterogeneousBlock}(C, (D_{i+1}, R_{i+1}))$. Thus we need a procedure that is similar to $\mathbf{HeterogeneousBlock}(C, (D_{i+1}, R_{i+1}))$ but instead of using the demands corresponding to two wavelength interchangers that each service two demands, it will use one such set of demands, which must exist by Lemma 16, and one single demand in D^H that uses input wavelength λ_1 and output wavelength γ_2 . Before presenting the new construction we need to prove that such a demand in D^H exists.

LEMMA 19. *Let C be a strictly nonblocking $k_1 \times k_2$ WDM split cross-connect where $(n_1 - 1)k_1 < k_2 < n_1k_1$. If D_i is a standard set of demands, then at least one demand in D^H has input wavelength λ_1 and output wavelength γ_2 .*

Proof. Since D_i is a standard set of demands, there must be one demand in D_i with output wavelength γ_2 for each output fiber in \mathcal{O}^H . The set of all such demands is by definition D^H . If $(n_1 - 1)k_1 < k_2 < n_1k_1$, then $z = n_1k_1 - k_2 < k_1$. The total number of demands is $k_2 + z = n_1k_1$. Therefore every input fiber has a demand for every input wavelength. Only z input fibers have demands with input wavelength λ_1 in D^A . Therefore at least one input fiber must have n_1 demands originating from it, all of which are in D^H . Since no two demands from the same input fiber can have the same input wavelength, this input fiber must have exactly one demand for each possible input wavelength. Therefore it has a demand in D^H with input wavelength λ_1 . Since all demands in D^H have output wavelength γ_2 , this demand is a demand in D^H with input wavelength λ_1 and output wavelength γ_2 . \square

Now assume we have a standard set of demands D_i and a standard routing R_i that routes D_i in such a way that all $n_1k_1 - 1$ wavelength interchangers in C service a demand in D_i . We can use Lemma 19 to show that in fact C must have n_1k_1 wavelength interchangers if it is strictly nonblocking.

First, we need to define the following notation. Let p_1 and p_2 be paths such that the last node v in p_1 is the first node in p_2 . We will use the notation $p_1||p_2$ to denote the path formed by following p_1 to v and then following p_2 . Consider $\mathbf{HeterogeneousBlock2}(C, (D_i, R_i))$, defined as follows.

HeterogeneousBlock2($C, (D_i, R_i)$).

1. Take one wavelength interchanger, WI_v , in W_i^B that services two demands and one wavelength interchanger, WI_u , in W^H that services a demand in D^H with input wavelength λ_1 and output wavelength γ_2 .
2. **Uncross**(WI_v).
3. Let $d_{v1} = (a_{v1}, \lambda_1, b_{v1}, \gamma_1)$ and $d_{v2} = (a_{v2}, \lambda_2, b_{v2}, \gamma_2)$ be the two resulting demands that WI_v services.
4. Let $d_u = (I_u, \lambda_1, h_2, \gamma_2)$ be the demand that WI_u services.
5. Remove d_u from D_i and replace it with demand $d'_u = (I_u, \lambda_1, h_2, \gamma_1)$ and route d'_u along the path on which d_u was routed.

6. Remove $d_{v1} = (a_{v1}, \lambda_1, b_{v1}, \gamma_1)$ and route all remaining demands according to R_i .
7. Add the demand $d'_{v1} = (a_{v1}, \lambda_1, h_2, \gamma_2)$ to D_i , add a valid route r'_{v1} for d'_{v1} to R_i , and let p'_{v1} be the input path of r'_{v1} .
8. Let WI_x be the wavelength interchanger in W_i^F that services $d'_{v1} = (a_{v1}, \lambda_1, h_2, \gamma_2)$.
9. Let $d_x = (a_{x2}, \lambda_2, b_{x1}, \gamma_1)$ be the other demand that WI_x services and let p_{x1} be the output path of the route for d_x .
10. Remove d_x and d'_{v1} from D_i .
11. Add $d_q = (a_{v1}, \lambda_1, b_{x1}, \gamma_1)$ to D_i to create D_{i+1} and add the valid route $r_q = p'_{v1} || p_{x1}$ for d_x to R_i to create R_{i+1} .
12. For any demand $d_j \in D_{i+1}$ of the form $(I_e, \lambda_t, h_j, \gamma_2)$, remove d_j from D_{i+1} , replace it with the demand $d'_j = (I_e, \lambda_t, h_j, \gamma_1)$, and route d'_j along the same path that d_j was routed on.
13. Add $(a_{x2}, \lambda_2, b_{v1}, \gamma_1)$ to D_{i+1} and add a valid route for this demand to R_{i+1} .
14. For any demand $d'_j \in D_{i+1}$ of the form $(I_e, \lambda_t, h_j, \gamma_1)$, remove d'_j from D_{i+1} , replace it with demand $d_j = (I_e, \lambda_t, h_j, \gamma_2)$, and route d_j along the same path that d'_j was routed on.
15. Return (D_{i+1}, R_{i+1}) .

Figure 15 shows the steps of **HeterogeneousBlock2** $(C, (D_i, R_i))$. In Lemma 20 we prove its correctness.

LEMMA 20. *If C is a strictly nonblocking $k_1 \times k_2$ WDM split cross-connect with $n_1 k_1 - 1$ wavelength interchangers where $(n_1 - 1)k_1 < k_2 < n_1 k_1$, D_i is a standard set of demands, and R_i is a standard routing of the demands in D_i that uses all $n_1 k_1 - 1$ wavelength interchangers, then **HeterogeneousBlock2** $(C, (D_i, R_i))$ can be executed and $(D_{i+1}, R_{i+1}) = \mathbf{HeterogeneousBlock2}(C, (D_i, R_i))$, where D_{i+1} is a standard set of demands and R_{i+1} is a standard routing of the demands in D_{i+1} such that all $n_1 k_1 - 1$ wavelength interchangers in C service a demand in D_{i+1} . Also, D_{i+1} and R_{i+1} are such that $|W^H|$ remains fixed, $|W_{i+1}^B| = |W_i^B| + 1$, and $|W_{i+1}^F| = |W_i^F| - 1$.*

Proof. We look at each step of **HeterogeneousBlock2** $(C, (D_i, R_i))$ and prove its correctness. Consider first Steps 1–4.

1. Take one wavelength interchanger, WI_v , in W_i^B that services two demands and one wavelength interchanger WI_u in W^H that services a demand in D^H with input wavelength λ_1 and output wavelength γ_2 .
2. **Uncross** (WI_v) .
3. Let $d_{v1} = (a_{v1}, \lambda_1, b_{v1}, \gamma_1)$ and $d_{v2} = (a_{v2}, \lambda_2, b_{v2}, \gamma_2)$ be the two resulting demands that WI_v services.
4. Let $d_u = (I_u, \lambda_1, h_2, \gamma_2)$ be the demand that WI_u services.

See Figure 16.

Lemmas 16 and 19 show the existence of a wavelength interchanger $WI_v \in W_i^B$ that services two demands and a wavelength interchanger $WI_u \in W^H$ that services a demand with input wavelength λ_1 and output wavelength γ_2 . After “uncrossing” the demands that WI_v services, we change the output wavelength from γ_2 to γ_1 of the demand d_u that WI_u services. The inductive assumption that R_i is a standard routing allows us to change this output wavelength without rerouting the demand.

By the definition of a standard set of demands, no other demand in D_i used the output fiber h_2 that d_u uses. Thus consider Step 5.

5. Remove d_u from D_i and replace it with demand $d'_u = (I_u, \lambda_1, h_2, \gamma_1)$ and route d'_u along the path on which d_u was routed.

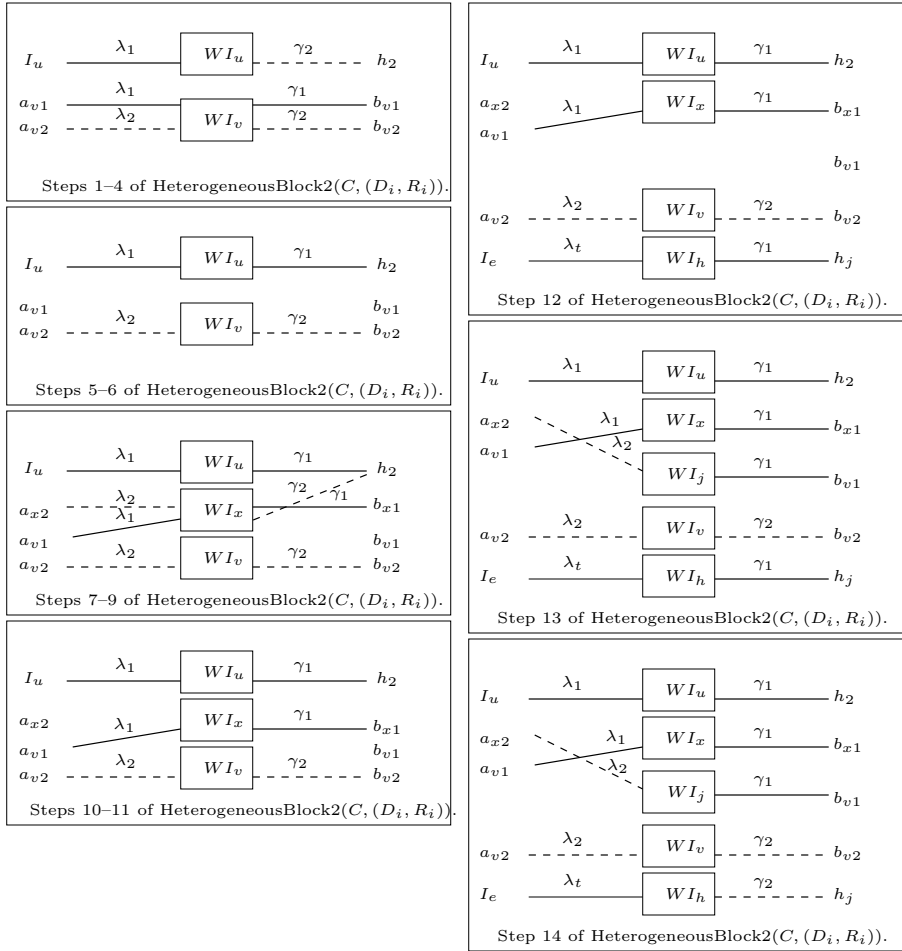


FIG. 15. The steps in $HeterogeneousBlock2(C, (D_i, R_i))$.

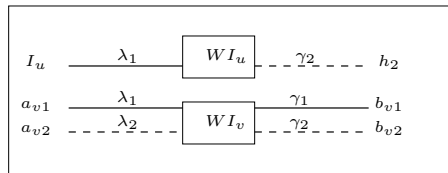


FIG. 16. Steps 1-4 of $HeterogeneousBlock2(C, (D_i, R_i))$.

After Step 5, when we change the only demand in D_i that uses h_2 to a demand with output wavelength γ_1 , it becomes valid to add a new demand with output wavelength γ_2 and output fiber h_2 . After Step 6, input fiber a_{v1} will have input wavelength λ_1 free.

6. Remove $d_{v1} = (a_{v1}, \lambda_1, b_{v1}, \gamma_1)$ and route all remaining demands according to R_i .

See Figure 17.

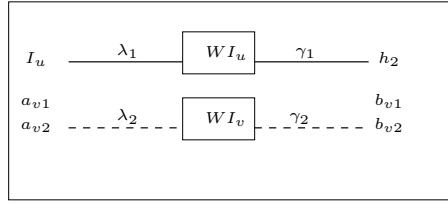


FIG. 17. Steps 5 and 6 of $HeterogeneousBlock2(C, (D_i, R_i))$.

We then add the demand, from input fiber a_{v1} to output fiber h_2 ; see Figure 18. This new demand uses input wavelength λ_1 and output wavelength γ_2 . Therefore it must be serviced by a wavelength interchanger $WI_x \in W_i^F$ since all of the wavelength interchangers in W_i^B and W^H are already servicing a demand with either input wavelength λ_1 or output wavelength γ_2 . The second demand d_x that WI_x services must exist because every wavelength interchanger in C services a demand in D_i . Thus Steps 9 and 10 are as follows.

7. Add the demand $d'_{v1} = (a_{v1}, \lambda_1, h_2, \gamma_2)$ to D_i , add a valid route r'_{v1} for d'_{v1} to R_i , and let p'_{v1} be the input path of r'_{v1} .

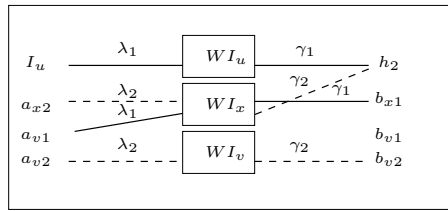


FIG. 18. Steps 7–9 of $HeterogeneousBlock2(C, (D_i, R_i))$.

9. Let $d_x = (a_{x2}, \lambda_2, b_{x1}, \gamma_1)$ be the other demand that WI_x services and let p_{x1} be the output path of the route for d_x .

10. Remove d_x and d'_{v1} from D_i .

Since R_i is able to route the first half of demand d'_v along p'_{v1} with wavelength λ_1 and the second half of d_x along p_{x1} with wavelength γ_1 , it must be possible to route demand d_q from Step 11 along $r_q = p'_{v1} || p_{x1}$; see Figure 19.

11. Add $d_q = (a_{v1}, \lambda_1, b_{x1}, \gamma_1)$ to D_i to create D_{i+1} and add the valid route $r_q = p'_{v1} || p_{x1}$ for d_x to R_i to create R_{i+1} .

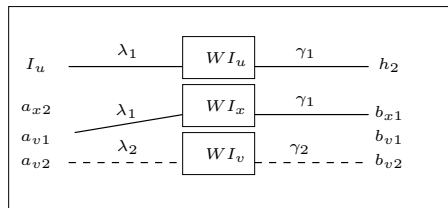


FIG. 19. Steps 10 and 11 of $HeterogeneousBlock2(C, (D_i, R_i))$.

Notice that this adds WI_x to W_{i+1}^B .

Next in Step 12, we “switch” the output wavelength used by all demands in D^H ; see Figure 20. Since R_i is a standard routing, switching the output wavelength of

the demands in D^H does not make their routes invalid. The reason for switching the output wavelength of all demands in D^H is to ensure that the new demand in Step 13, is not serviced by a wavelength interchanger in W^H ; see Figure 21.

12. For any demand $d_j \in D_{i+1}$ of the form $(I_e, \lambda_t, h_j, \gamma_2)$, remove d_j from D_{i+1} , replace it with the demand $d'_j = (I_e, \lambda_t, h_j, \gamma_1)$, and route d'_j along the same path that d_j was routed on.

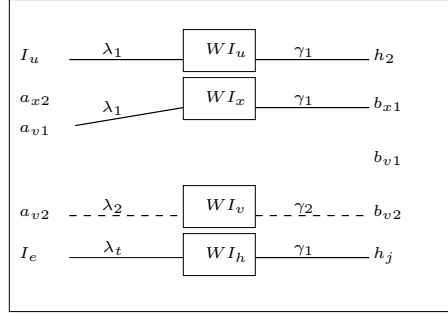


FIG. 20. Step 12 of $HeterogeneousBlock2(C, (D_i, R_i))$.

13. Add $(a_{x2}, \lambda_2, b_{v1}, \gamma_1)$ to D_{i+1} and add a valid route for this demand to R_{i+1} .

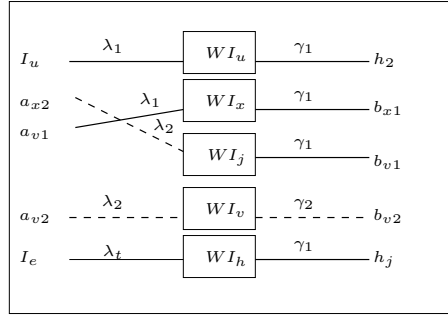


FIG. 21. Step 13 of $HeterogeneousBlock2(C, (D_i, R_i))$.

Since this new demand is a valid demand and C is strictly nonblocking, there must exist a valid route for this demand. Furthermore, the output path of the route for the new demand must be edge-disjoint from all demands that use output wavelength γ_1 . Therefore the route chosen for the new demand will maintain the inductive invariant that all demands not in D^H use edge-disjoint output paths from those used by demands in D^H . After we have found a route for the demand made in Step 13, Step 14 switches the output wavelength of all demands in D^H back to γ_2 so that D_{i+1} meets the definition of a standard set of demands; see Figure 22.

14. For any demand $d'_j \in D_{i+1}$ of the form $(I_e, \lambda_t, h_j, \gamma_1)$, remove d'_j from D_{i+1} , replace it with demand $d_j = (I_e, \lambda_t, h_j, \gamma_2)$, and route d_j along the same path that d'_j was routed on.

Notice that $|D_{i+1}| = k_2 + z$. Furthermore, $2z$ of the demands in D_{i+1} are demands with an input fiber in \mathcal{I}^A , an output fiber in \mathcal{O}^A , input wavelength either λ_1 or λ_2 , and output wavelength either γ_1 or γ_2 . The other $k_2 - z$ demands in D_{i+1} have output wavelength γ_2 and an output fiber in \mathcal{O}^H . Additionally, the routing R_{i+1} is such that each wavelength interchanger in W^H services at most one demand and

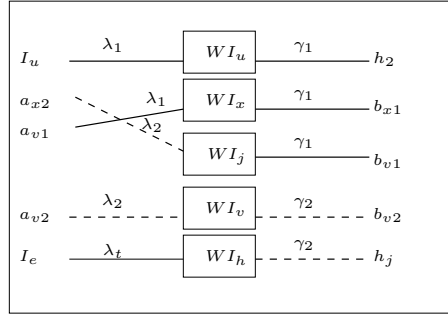


FIG. 22. Step 14 of $HeterogeneousBlock2(C, (D_i, R_i))$.

the set of output paths for demands in D^H are edge-disjoint from all output paths for demands in D_{i+1}^A . Therefore R_{i+1} is a standard routing of D_{i+1} . Furthermore, $|W^H|$ remains fixed, $|W_{i+1}^B| = |W_i^B| + 1$, and $|W_{i+1}^F| = |W_i^F| - 1$. Finally, since **HeterogeneousBlock2**($C, (D_i, R_i)$) only ever removed a demand from a wavelength interchanger that already serviced two demands, each wavelength interchanger in C must service a demand in D_{i+1} under R_{i+1} . \square

By Lemma 17 we can repeatedly perform **HeterogeneousBlock**($C, (D_i, R_i)$) until all $n_1 k_1 - 1$ wavelength interchangers are used to service a standard set of demands D_i routed according to a standard routing R_i of D_i . Using Lemma 20 we can then repeatedly augment D_i and R_i until we arrive at a standard set of demands D_t and a standard routing R_t of D_t for which $|W_t^F| = 0$. In **HeterogeneousBlock2**($C, (D_t, R_t)$), the new demand in Step 7 is a valid demand and therefore any strictly nonblocking WDM cross-connect must be able to find a route for this demand. However, since it must be serviced by a wavelength interchanger from W_t^F , C must not be strictly nonblocking. This contradiction leads to the following result.

THEOREM 21. *Any strictly nonblocking $k_1 \times k_2$ WDM split cross-connect with $(n_1 - 1)k_1 < k_2 < n_1 k_1$, where $n_1 > 1$ and $n_2 > 1$ are the number of available wavelengths in each input fiber and output fiber, respectively, will have at least $n_1 k_1$ wavelength interchangers.*

Together the results in Theorems 15, 18, and 21 imply the following.

THEOREM 22. *Any strictly nonblocking $k_1 \times k_2$ WDM split cross-connect with $n_1 > 1$ input wavelengths and $n_2 > 1$ output wavelengths must have at least $\min(k_1 + k_2 - 1, n_1 k_1)$ wavelength interchangers.*

8. Conclusions and future work. For a homogeneous $k \times k$ WDM cross-connect we have presented an optimal design using $2k - 1$ wavelength interchangers. For heterogeneous cross-connects we have considered only the case of a $k_1 \times k_2$ WDM split cross-connect and have shown that $\min(k_1 + k_2 - 1, n_1 k_1)$ wavelength interchangers are necessary and sufficient. One obvious open question is to consider whether a nonsplit design is optimal for heterogeneous cross-connects. The reason we cannot extend Theorem 4 to heterogeneous cross-connects is that we need to be able to remove $n - 1$ wavelength interchangers along a long path in some cases of heterogeneous cross-connects. However, as with the homogeneous case, we can be sure only to remove two wavelength interchangers. Therefore a new idea is required to prove that split cross-connects are optimal for heterogeneous cross-connects. Considering general demand models for heterogeneous cross-connects is also an interesting line of work.

Another open area of work is to consider wide-sense nonblocking cross-connects. In [5] it is shown that wide-sense nonblocking $k \times k$ WDM split cross-connects with $n \geq O(k^2)$ available wavelengths require at least $2k - 1$ wavelength interchangers. However, it is still open as to whether a split cross-connect is optimal for wide-sense nonblocking cross-connects. Furthermore, for wide-sense nonblocking $k \times k$ WDM cross-connects with $2 < n < O(k^2)$, the optimal number of wavelength interchangers is unknown even if only split cross-connects are considered.

Acknowledgment. We would like to thank the anonymous referee for helpful comments.

REFERENCES

- [1] L. A. BASSALYGO AND M. S. PINSKER, *Complexity of an optimal nonblocking switching network without reconstructions*, Problems Inform. Transmission, 9 (1974), pp. 64–66.
- [2] L. A. BASSALYGO AND M. S. PINSKER, *Asymptotically optimal networks for generalized rearrangeable switching and generalized switching without rearrangement*, Problemy Peredachi Informatsii, 16 (1980), pp. 94–98.
- [3] V. E. BENÈS, *Mathematical Theory of Connecting Networks and Telephone Traffic*, Academic Press, New York, NY, 1965.
- [4] P. FELDMAN, J. FRIEDMAN, AND N. PIPPENGER, *Wide-sense nonblocking networks*, SIAM J. Discrete Math., 1 (1988), pp. 158–173.
- [5] P. HAXELL, A. RASALA, G. WILFONG, AND P. WINKLER, *Wide-sense nonblocking WDM cross-connects*, in Proceedings of the ESA, Rome, Italy, 2002.
- [6] J. KLEINBERG AND A. KUMAR, *Wavelength conversion in optical networks*, in Proceedings of the Tenth Annual SODA, Baltimore, ACM, New York, 1999, pp. 566–575.
- [7] N. PIPPENGER, *Telephone switching networks*, in Proceedings of the AMS Symposium on Applied Mathematics, 1982, pp. 101–133.
- [8] N. PIPPENGER AND L. VALIANT, *Shifting graphs and their applications*, J. Assoc. Comput. Math., 23 (1976), pp. 423–432.
- [9] R. RAMASWAMI AND G. H. SASAKI, *Multiwavelength optical networks with limited wavelength conversion*, in Proceedings of the IEEE INFOCOM, Kobe, Japan, 2, 1997, pp. 489–498.
- [10] A. RASALA AND G. WILFONG, *Strictly nonblocking WDM cross-connects for heterogeneous networks*, in Proceedings of the STOC, Portland, OR, 2000.
- [11] A. RASALA AND G. WILFONG, *Strictly nonblocking WDM cross-connects*, in Proceedings of the SODA, 2000, San Francisco, CA, pp. 606–615.
- [12] C. E. SHANNON, *Memory requirements in a telephone exchange*, Bell System Tech. J., 29 (1950), pp. 343–349.
- [13] G. WILFONG, B. MIKKELSEN, C. DOERR, AND M. ZIRNGIBL, *WDM cross-connect architectures with reduced complexity*, Journal of Lightwave Technology, (1999), pp. 1732–1741.
- [14] G. WILFONG AND P. WINKLER, *Ring routing and wavelength translation*, in Proceedings of the SODA, San Francisco, ACM, New York, 1998, pp. 333–341.

STRONG SPATIAL MIXING WITH FEWER COLORS FOR LATTICE GRAPHS*

LESLIE ANN GOLDBERG[†], RUSSELL MARTIN[‡], AND MIKE PATERSON[†]

Abstract. Recursively-constructed couplings have been used in the past for mixing on trees. We show how to extend this technique to nontree-like graphs such as lattices. Using this method, we obtain the following general result. Suppose that G is a triangle-free graph and that for some $\Delta \geq 3$, the maximum degree of G is at most Δ . We show that the spin system consisting of q -colorings of G has strong spatial mixing, provided $q > \alpha\Delta - \gamma$, where $\alpha \approx 1.76322$ is the solution to $\alpha^\alpha = e$, and $\gamma = \frac{4\alpha^3 - 6\alpha^2 - 3\alpha + 4}{2(\alpha^2 - 1)} \approx 0.47031$. Note that we have no additional lower bound on q or Δ . This is important for us because our main objective is to have results which are applicable to the lattices studied in statistical physics, such as the integer lattice \mathbb{Z}^d and the triangular lattice. For these graphs (in fact, for any graph in which the distance- k neighborhood of a vertex grows subexponentially in k), strong spatial mixing implies that there is a unique infinite-volume Gibbs measure. That is, there is one macroscopic equilibrium rather than many. Our general result gives, for example, a “hand proof” of strong spatial mixing for 7-colorings of triangle-free 4-regular graphs. (Computer-assisted proofs of this result were provided by Salas and Sokal [*J. Stat. Phys.*, 86 (1997), pp. 551–579] (for the rectangular lattice) and by Buble, Dyer, Greenhill, and Jerrum [*SIAM J. Comput.*, 29 (1999), pp. 387–400].) It also gives a hand proof of strong spatial mixing for 5-colorings of triangle-free 3-regular graphs. (A computer-assisted proof for the special case of the hexagonal lattice was provided earlier by Salas and Sokal [*J. Stat. Phys.*, 86 (1997), pp. 551–579].) Toward the end of the paper we show how to improve our general technique by considering the geometry of the lattice. The idea is to construct the recursive coupling from a system of recurrences rather than from a single recurrence. We use the geometry of the lattice to derive the system of recurrences. This gives us an analysis with a horizon of more than one level of induction, which leads to improved results. We illustrate this idea by proving strong spatial mixing for $q = 10$ on the lattice \mathbb{Z}^3 . Finally, we apply the idea to the triangular lattice, adding computational assistance. This gives us a (machine-assisted) proof of strong spatial mixing for 10-colorings of the triangular lattice. (Such a proof for 11 colors was given by Salas and Sokal [*J. Stat. Phys.*, 86 (1997), pp. 551–579].) For completeness, we also show that our strong spatial mixing proof implies rapid mixing of Glauber dynamics for sampling proper colorings of neighborhood-amenable graphs. (It is known that strong spatial mixing often implies rapid mixing, but existing proofs seem to be written for \mathbb{Z}^d .) Thus our strong spatial mixing results give rapid mixing corollaries for neighborhood-amenable graphs such as lattices.

Key words. strong spatial mixing, proper graph coloring, antiferromagnetic Potts model

AMS subject classifications. 60J20, 68W20

DOI. 10.1137/S0097539704445470

1. Introduction. This paper is concerned with (proper) colorings of an infinite graph G , such as the integer lattice \mathbb{Z}^d . A coloring is an assignment of colors from the set $\{1, \dots, q\}$ to the vertices. It is *proper* if adjacent vertices receive different colors. Proper colorings correspond to configurations in the *zero-temperature antiferromagnetic Potts model*. Two important closely-related questions which have received a lot of recent attention follow.

*Received by the editors August 2, 2004; accepted for publication (in revised form) April 29, 2005; published electronically November 18, 2005. This work was partially supported by the EPSRC grant “Discontinuous Behavior in the Complexity of Randomized Algorithms.” A preliminary version of this paper appeared in *Proceedings of the 45th Annual IEEE Symposium on Foundations of Computer Science (FOCS 2004)*, Rome, pp. 562–571.

<http://www.siam.org/journals/sicomp/35-2/44547.html>

[†]Department of Computer Science, University of Warwick, Coventry, CV4 7AL, UK.

[‡]Department of Computer Science, University of Liverpool, Liverpool, L69 7ZF, UK.

- Do boundary effects decay exponentially? (This notion is known as “strong spatial mixing.”)
- Is there a unique infinite-volume Gibbs measure? (The converse situation is often called a “phase transition.”)

See Weitz’s Ph.D. thesis [31] and Martinelli’s lecture notes [22] for an exposition of this material and the papers [1, 3, 13, 20, 23, 27] for some recent (and not so recent) results. For graphs like regular lattices (in fact, for any graph in which the distance- k neighborhood of a vertex grows subexponentially in k), strong spatial mixing implies that there is a unique infinite-volume Gibbs measure; see [31] and [22] for details.¹ For these graphs, the two questions above are also known to be closely related to a third question:

- Are Glauber dynamics rapidly mixing on finite pieces of the graph?

For graphs such as lattice graphs, strong spatial mixing implies rapid mixing; more details are given in section 7. A number of papers have given bounds on the number of colors that are necessary for rapid mixing, both for general graphs [9, 10, 14, 16, 17, 19, 25, 29] and for specific graphs and lattices [1, 15, 21, 24].

1.1. Definitions and background. In order to define “strong spatial mixing” and “infinite-volume Gibbs measure,” we need notation for describing colorings of finite regions of the infinite graph G . A *region* R of G is a (not necessarily connected) subset of the vertices. A *coloring* of R is a function from R to the set of *colors* $Q = \{1, \dots, q\}$. If R is nonempty and finite, then ∂R denotes the vertex boundary around R . That is, ∂R is the set of vertices that are not in R , but are adjacent to R . A coloring of ∂R is a function from ∂R to the set $\{0\} \cup Q$. The color “0” corresponds to an unconstrained boundary vertex. Given a coloring \mathcal{B} of ∂R , a coloring C of R is said to be *proper* if adjacent vertices in R receive different colors, and vertices in R receive colors different from adjacent boundary vertices. $S(\mathcal{B})$ denotes the set of proper colorings of R and $\pi_{\mathcal{B}}$ denotes the uniform distribution on $S(\mathcal{B})$. For any $\Lambda \subseteq R$, $\pi_{\mathcal{B}, \Lambda}$ denotes the distribution on colorings of Λ induced by $\pi_{\mathcal{B}}$.

A measure μ on the set of proper colorings of G is an infinite-volume *Gibbs measure* (with respect to the uniform specification) if, for any finite region R , the conditional probability distribution $\mu(\cdot \mid \sigma_{\bar{R}})$ (conditioned on the coloring $\sigma_{\bar{R}}$ of all vertices other than those in R) is the uniform distribution on proper colorings of R . Infinite-volume Gibbs measures exist for any G . The problem of determining whether there is more than one infinite-volume Gibbs measure for a given “specification” is known as the DLR problem (for Dobrushin, Lanford, and Ruelle) in statistical physics (see [3]).

An important notion in statistical physics is whether the system (as specified by the finite-volume Gibbs measures) satisfies *strong spatial mixing* [22]. Informally, this means that for any finite set of vertices R , if you consider two different colorings \mathcal{B} and \mathcal{B}' of the boundary of R which differ at a single vertex y , then the effect that this difference has on a subset $\Lambda \subseteq R$ decays exponentially with the distance from Λ to y . For the formal definition (which we take from [13]), recall that the *total variation*

¹The formal definition of “strong spatial mixing” that we use [13, 22] requires that there be exponential decay in the effect of a single discrepancy at the boundary of a region. If the graph has subexponential growth (e.g., the distance- k neighborhood of a vertex grows subexponentially in k), then this implies uniqueness.

distance between distributions θ_1 and θ_2 on Ω is

$$d_{\text{tv}}(\theta_1, \theta_2) = \frac{1}{2} \sum_{i \in \Omega} |\theta_1(i) - \theta_2(i)| = \max_{A \subseteq \Omega} |\theta_1(A) - \theta_2(A)|.$$

We can now define strong spatial mixing.

DEFINITION 1. *The spin system specified by uniform finite-volume Gibbs measures on proper q -colorings of G has strong spatial mixing if there are constants β and $\beta' > 0$ such that for any nonempty finite region R , any $\Lambda \subseteq R$, any vertex $y \in \partial R$, and any pair of colorings $(\mathcal{B}, \mathcal{B}')$ of ∂R which differ only at y ,*

$$d_{\text{tv}}(\pi_{\mathcal{B}, \Lambda}, \pi_{\mathcal{B}', \Lambda}) \leq \beta |\Lambda| \exp(-\beta' d(y, \Lambda)),$$

where $d(y, \Lambda)$ is the distance within R from the vertex y to the region Λ .

For a wide family of graphs, this notion of strong spatial mixing implies that there is a unique infinite-volume Gibbs measure with exponentially decaying correlations. For further details on this connection, see [22, 30, 31].

In order to demonstrate that there is strong spatial mixing for the systems studied in this paper, we will consider an arbitrary finite set of vertices R and two different colorings \mathcal{B} and \mathcal{B}' of the boundary of R which differ at a single vertex y . We will show inductively that there is a coupling of the two conditional distributions in which, for every vertex $v \in R$, the probability of disagreement at v is exponentially small (as a function of its distance to the boundary discrepancy).

Another issue that we address is the mixing time of Glauber dynamics for *sampling* proper graph colorings. Let R be a finite region and let \mathcal{B} be a coloring of ∂R . The (heat-bath) Glauber dynamics is a Markov chain that can be used to sample from $S(\mathcal{B})$, the set of proper colorings that are consistent with the coloring \mathcal{B} of ∂R . The transition from a coloring $\sigma \in S(\mathcal{B})$ is made by choosing a vertex v uniformly at random from R and then recoloring v from the conditional distribution induced by the colors of the neighbors of v .

A condition sufficient for the Glauber dynamics Markov chain to be *connected* (i.e., any proper coloring can be obtained from another proper coloring by a series of the transitions described) is to have $q \geq \Delta + 2$, where Δ is the maximum degree of the graph. The stationary distribution of this Markov chain is $\pi_{\mathcal{B}}$, the uniform distribution on $S(\mathcal{B})$. In this setting, the question of interest is to determine the *mixing time*, $\tau(\delta)$, of the Glauber dynamics chain defined as

$$\tau(\delta) = \min\{t : d_{\text{tv}}(P^{(t)}(\sigma, \cdot), \pi_{\mathcal{B}}) \leq \delta \quad \forall t' \geq t\}.$$

Here $P^{(t)}(\sigma, \nu)$ is the probability of moving from σ to ν in exactly t steps of the Markov chain.

Heat-bath dynamics on larger regions is defined similarly except that a “block” of K vertices is updated during each transition; see [13] for one example of heat-bath dynamics on the lattice \mathbb{Z}^2 . We discuss a general version of heat-bath dynamics later when we examine the connections between strong spatial mixing and rapid mixing more closely.

For some graphs with subexponential growth (that is, for graphs in which the volume of increasing balls around any vertex increases subexponentially with the radius), it is well known that strong spatial mixing implies rapid mixing of Glauber dynamics. For example, [13] provides a purely combinatorial proof that when G is the

d -dimensional integer lattice \mathbb{Z}^d , if the system has strong spatial mixing, then there exists a finite integer K for which the heat-bath dynamics on a “cube” of side length K mixes in $O(n \log n)$ time, where $n = |R|$. This result holds for the “permissive” case, which corresponds to the restriction $q > \Delta + 1$ in our setting. As [13] observes, it is also known for \mathbb{Z}^d that strong spatial mixing implies $O(n \log n)$ mixing for Glauber dynamics (see [6, 22]) though no purely combinatorial proof of this fact is known. Also, the proofs as written may need to be modified to apply to the “zero-temperature” (proper coloring) case. Even without using these results in the zero-temperature case, we can deduce that Glauber dynamics mix in polynomial time (in fact, in $O(n^2)$ time) for a general family of graphs. This can be shown by using the comparison method of Diaconis and Saloff-Coste [7] to turn a rapid mixing result for heat-bath dynamics (for a fixed K) into a rapid mixing result for Glauber dynamics. For an example on the integer lattice \mathbb{Z}^2 , refer to Theorem 2 of [1] which shows rapid mixing for Glauber dynamics on 6-colorings of square pieces of \mathbb{Z}^2 . (For convenience, the authors have bounded the mixing time as $O(n^2 \log n)$, but if one wanted to tighten the bound to $O(n^2)$ by tuning the parameters in the comparison, this is possible; see, for example, [11, Example 9].)

The theorems in [13] are explicitly stated for the integer lattice \mathbb{Z}^d , but the authors state that similar techniques apply to any lattice with subexponential growth. This is mentioned as a footnote in [13] and is discussed more fully in [31]. We provide a proof that strong spatial mixing implies rapid mixing of Glauber dynamics for a class of graphs that we call neighborhood-amenable, whose definition is given below.

First, for any vertex $v \in G$ and a nonnegative integer d , let $Ball_d(v)$ denote the set of vertices that are at most distance d from v . Thus we have $Ball_0(v) = \{v\}$.

DEFINITION 2. For a nonnegative integer d , let $T_d = \sup_{v \in G} \frac{|\partial Ball_d(v)|}{|Ball_d(v)|}$. G is said to be neighborhood-amenable if $\inf_d T_d = 0$.

Neighborhood-amenability is a related, yet different, notion to amenability in graphs.² From the definition we can see that for a neighborhood-amenable graph, given any real number $c > 0$, we can find $d \geq 0$ such that $\frac{|\partial Ball_d(v)|}{|Ball_d(v)|} \leq c$, uniformly in v , meaning that the “surface-area-to-volume” ratio of balls can be made arbitrarily small with a suitable choice of radius d . Most natural lattices, such as the triangular lattice and \mathbb{Z}^k , are neighborhood-amenable.

Conditions under which we can prove rapid mixing of Glauber dynamics on neighborhood-amenable graphs are given in Theorem 8 in section 1.3.

1.2. The framework. Our results rely on considering proper colorings of a finite region for a pair of boundary colorings of that region that differ on the color of a single vertex. First, we outline the general framework in which we operate.

Let G denote an infinite graph with maximum degree Δ . Let R be a *finite* subgraph of G , and as before, define ∂R to be the boundary of R , i.e., those vertices in G that are not in R but are joined by an edge to at least one vertex of R . First we give the following definition.

DEFINITION 3. A vertex-boundary pair X consists of

- a nonempty finite region R_X of the graph G ,
- a distinguished vertex v_X in ∂R_X , and

²An infinite graph G is *amenable* if

$$\inf \left\{ \frac{|\partial S|}{|S|} : S \text{ is a finite and nonempty subset of } V(G) \right\} = 0.$$

- a pair $(\mathcal{B}_X^1, \mathcal{B}_X^2)$ of colorings of ∂R_X (using the colors $Q \cup \{0\}$) which differ only on the vertex v_X . We require that the two colors $\mathcal{B}_X^1(v_X)$ and $\mathcal{B}_X^2(v_X)$ are both in the set Q . That is, the two boundary colorings differ on the color of v_X , but this vertex is not an unconstrained vertex (with color 0) in either boundary coloring.

We are interested in the effect that the difference in color at v_X has on the other vertices. Let $S(\mathcal{B}_X^1)$ be the set of proper q -colorings of R_X that are consistent with the boundary coloring \mathcal{B}_X^1 , and similarly define $S(\mathcal{B}_X^2)$. We use $\pi_{\mathcal{B}_X^1}$ (resp., $\pi_{\mathcal{B}_X^2}$) to denote the uniform distribution on $S(\mathcal{B}_X^1)$ (resp., $S(\mathcal{B}_X^2)$).

We want to construct a coupling Ψ_X of the distributions $\pi_{\mathcal{B}_X^1}$ and $\pi_{\mathcal{B}_X^2}$, i.e., a joint distribution on $S(\mathcal{B}_X^1) \times S(\mathcal{B}_X^2)$ that has $\pi_{\mathcal{B}_X^1}$ and $\pi_{\mathcal{B}_X^2}$ as its marginal distributions. For such a coupling Ψ_X and for each vertex $f \in R_X$, we define the indicator random variable $1_{\Psi_X, f}$ for the event that, when a pair of colorings is drawn according to Ψ_X , the color of f differs in these two colorings. We would like to show that $\sum_{f \in R_X} \mathbb{E}[1_{\Psi_X, f}]$ is small. If this quantity is small enough for all vertex-boundary pairs X , we can use that conclusion to infer strong spatial mixing. We can also show rapid mixing of Glauber dynamics for a general class of graphs. One way to show that the sum is small is to show that $\mathbb{E}[1_{\Psi_X, f}]$ decreases rapidly as the distance between v_X and f grows. We give a method to construct a coupling using couplings of subgraphs which may overlap. In the course of the proof we use what we call an ε -coupling cover for G , whose definition follows.

DEFINITION 4. *Let G denote an infinite graph with maximum degree Δ . Fix $\varepsilon > 0$. We say that G has an ε -coupling cover if for all vertex-boundary pairs X , there is a coupling Ψ_X of $\pi_{\mathcal{B}_X^1}$ and $\pi_{\mathcal{B}_X^2}$ such that*

$$\sum_{f \in R_X} \mathbb{E}[1_{\Psi_X, f}] \leq \frac{\Delta}{\varepsilon}.$$

Thus, if G has an ε -coupling cover, then the sum $\sum_{f \in R_X} \mathbb{E}[1_{\Psi_X, f}]$ is small. The precise manner in which this property is used to prove strong spatial mixing is described in section 4 after we lay the groundwork in sections 2 and 3.

1.3. Our results. Our first main result is the following theorem.

THEOREM 5. *Let α be the solution to $\alpha^\alpha = e$ (so $\alpha \approx 1.76322$), and $\gamma = \frac{4\alpha^3 - 6\alpha^2 - 3\alpha + 4}{2(\alpha^2 - 1)} \approx 0.47031$. Let G denote an infinite triangle-free graph, and suppose that for some $\Delta \geq 3$ the maximum degree of G is at most Δ . The spin system specified by uniform finite-volume Gibbs measures on proper q -colorings of G has strong spatial mixing if $q > \alpha\Delta - \gamma$.*

With some additional consideration of the structure of the graph we can prove two additional results about strong spatial mixing. Theorem 6, proven in section 5, uses a system of recurrence relations to show strong spatial mixing. The geometry of \mathbb{Z}^3 plays an important role in deriving this system of recurrences. This special case is not covered by our general result above since $\alpha \cdot 6 - \gamma \approx 10.10901$.

THEOREM 6. *The spin system specified by uniform finite-volume Gibbs measures on proper 10-colorings of \mathbb{Z}^3 has strong spatial mixing.*

With computational assistance, we also show another special case that is not covered by our general theorem. (This graph has lots of triangles in it!) In this case we again derive a system of recurrence relations to show strong spatial mixing; see section 6 for more details.

THEOREM 7. *The spin system specified by uniform finite-volume Gibbs measures on proper 10-colorings of the triangular lattice has strong spatial mixing.*

In addition to the results on strong spatial mixing, we prove a general result on rapid mixing of Glauber dynamics for sampling proper colorings. Provided there exists an ε -coupling cover, Glauber dynamics is rapidly mixing for neighborhood-amenable graphs.

THEOREM 8. *Let G denote an infinite neighborhood-amenable graph with maximum degree Δ . Let R be a finite subgraph of G with $|R| = n$ and $\mathcal{B}(R)$ denoting a coloring of $\partial(R)$ using the colors $Q \cup \{0\}$. (We assume that $q \geq \Delta + 2$.)*

Suppose there exists $\varepsilon > 0$ such that G has an ε -coupling cover. Then the Glauber dynamics Markov chain on $S(\mathcal{B}(R))$ is rapidly mixing and $\tau(\delta) \in O(n(n + \log \frac{1}{\delta}))$.

Path coupling is used to prove this theorem. To show Theorem 8 we first examine a heat-bath Markov chain on “windows” in the graph (more specifically, small regions of the form $Ball_d(v) \cap R$ for a suitable d) and prove this chain mixes in time $O(n \log n)$. Then using the standard technique of comparing Markov chains, we are able to conclude that the simpler Glauber dynamics (single-vertex) chain is rapidly mixing in time $O(n^2)$. It is for this reason that we require an ε -coupling cover for G , since in our analysis we examine proper colorings of R that differ at a single vertex and we need to determine what happens in a single step of the heat-bath chain.

A brief review of path coupling, the comparison method, and all the details of the proof of Theorem 8 can be found in section 7.

In the proof of Theorem 5 we construct an ε -coupling cover (see Lemmas 15 and 20). Taking these lemmas and Theorem 8 together, we obtain the following corollary regarding rapid mixing of Glauber dynamics; see section 7.5 for some remarks on the “neighborhood-amenable” condition in the hypothesis of Theorem 8 and in the corollary.

COROLLARY 9. *Let G denote an infinite triangle-free, neighborhood-amenable graph and suppose that for some $\Delta \geq 3$ the maximum degree of G is at most Δ . Suppose $q > \alpha\Delta - \gamma$. Let R be a finite subgraph of G with $|R| = n$ and $\mathcal{B}(R)$ denoting a coloring of $\partial(R)$ using the colors $Q \cup \{0\}$. (We assume that $q \geq \Delta + 2$.)*

The Glauber dynamics chain on proper colorings of R compatible with $\mathcal{B}(R)$ is rapidly mixing with $\tau(\delta) \in O(n(n + \log \frac{1}{\delta}))$.

Since \mathbb{Z}^3 and the triangular lattice are neighborhood-amenable graphs, we also obtain the following corollaries. We note that in the course of showing strong spatial mixing for 10-colorings of \mathbb{Z}^3 and the triangular lattice, we prove the existence of an ε -coupling cover in each case. These results, together with Theorem 8, give us the two corollaries below; see sections 5 and 6, respectively, for more details on the existence of an ε -coupling cover in each case.

COROLLARY 10. *Let R denote a finite subgraph of \mathbb{Z}^3 with $|R| = n$. Let $\mathcal{B}(R)$ denote a coloring of ∂R with the colors $\{1, \dots, 10\} \cup \{0\}$; Glauber dynamics is rapidly mixing on the set of 10-colorings compatible with $\mathcal{B}(R)$ and $\tau(\delta) \in O(n(n + \log \frac{1}{\delta}))$.*

COROLLARY 11. *Let R denote a finite subgraph of the triangular lattice with $|R| = n$. Let $\mathcal{B}(R)$ denote a coloring of ∂R with the colors $\{1, \dots, 10\} \cup \{0\}$; Glauber dynamics is rapidly mixing on the set of 10-colorings compatible with $\mathcal{B}(R)$ and $\tau(\delta) \in O(n(n + \log \frac{1}{\delta}))$.*

1.4. Related work. Previous papers [20, 23, 24] have used recursively-constructed couplings to show rapid mixing and exponential decay of correlations on trees. To apply this approach more generally (i.e., to graphs other than trees) we need a mechanism for constructing a coupling from couplings of subgraphs (even

though these subgraphs may overlap). Our approach (see Lemma 12) gives an upper bound on the effect of the discrepancy at a site by summing over discrepancies at adjacent edges (using the triangle inequality as in path coupling). The subgraphs corresponding to these edges overlap but the triangle inequality is used a second time to bound the quality of the resulting coupling.

The closest directly applicable result similar to ours is that of Salas and Sokal [27]. They showed that strong spatial mixing occurs whenever $q > 2\Delta$. Given that strong spatial mixing and rapid mixing are sometimes interchangeable (as we noted above), it is perhaps more appropriate to compare our result with recent (stronger) results about rapid mixing for colorings. There are lots of these results. Since our goal is to have results which apply to lattices such as those studied in statistical physics (i.e., small Δ and small q) the most relevant result is the new theorem of Dyer et al. [10]. They show that if the girth of the graph is at least 5 (i.e., there are no 4-cycles or triangles) then Glauber dynamics is rapidly mixing provided $q > \max(\alpha\Delta, C)$, where C is an absolute constant (it depends upon $q - \alpha\Delta$ but not upon the number of vertices) which is at least 200. Our result (Theorem 5) can be viewed as a companion to that one. Both results apply when $q > \alpha\Delta$. Ours gives strong spatial mixing when the girth is at least 4 (and the maximum degree $\Delta \geq 3$). The result in [10] gives rapid mixing when the girth is at least 5 and $q \geq C$. The two results are interesting for different, but overlapping, classes of graphs. Ours is interesting (since it implies uniqueness of Gibbs measure and rapid mixing) even for graphs with very small degree (all the way down to $\Delta = 3$ and $q = 6$) but the applications to uniqueness and rapid mixing apply only if the graph is neighborhood-amenable (or some similar condition). (All natural lattices satisfy this.) The result of [10] is interesting even for graphs with other neighborhood growth properties, but it applies only if $q \geq C$.

Better results for rapid mixing are known when the degree, or the girth, is guaranteed to be large. (For graphs with large degree, the distribution is concentrated, so strong results are possible.) These results include rapid mixing for $q > \alpha\Delta$ assuming $\Delta = \Omega(\log n)$ and girth of at least 4 (Hayes and Vigoda [18]), rapid mixing for $q > \alpha\Delta$ assuming $\Delta = \Omega(\log n)$ and “local sparsity” (Frieze and Vera [14]), rapid mixing for $q > (1 + \varepsilon)\Delta$ assuming $\Delta = \Omega(\log n)$ and girth of at least 9 (Hayes and Vigoda [17]), and rapid mixing for graphs with girth of at least 6 when $q > \max(\beta\Delta, C')$ for some constant C' and $\beta \approx 1.49$ (Dyer et al. [10]).

Theorem 5 provides the first hand proof of strong spatial mixing for 7-colorings of triangle-free graphs with degree of at most 4. A machine-assisted proof for the rectangular lattice was provided by Salas and Sokal [27] and a machine-assisted proof of rapid mixing for triangle-free 4-regular graphs was provided by Buble, Dyer, Greenhill, and Jerrum [5]. Our result also shows strong spatial mixing for $q = 5$ and $\Delta = 3$. This is the first hand proof of strong spatial mixing for 5-colorings of triangle-free graphs with degree of at most 3. A machine-assisted proof for the special case of the hexagonal lattice was proved by Salas and Sokal [27]. They also give a machine-assisted proof for 4-colorings of this lattice.

In section 5 we show how to improve our general technique by considering the geometry of the lattice. The idea is to construct the recursive coupling from a system of recurrences rather than from a single recurrence. We use the geometry of the lattice to derive the system of recurrences. This gives us an analysis with a horizon of more than one level of induction, which leads to improved results. We illustrate this idea by proving strong spatial mixing for $q = 10$ on the lattice \mathbb{Z}^3 .

In section 6 we further extend our results using computational assistance. An idea used to reduce the amount of computation is the notion of a “relevant” boundary pair.

In order to reduce the search space, we want to look just at “relevant” boundary pairs, and not at all of them. Boundary pairs induced by vertex boundaries are “relevant,” and we can show by induction that our method recurses from relevant boundary pairs to relevant boundary pairs of subproblems. The proof of this fact again relies on the geometry of the lattice; see section 6 for details. Using the approach we obtain a (machine-assisted) proof of strong spatial mixing (and therefore, uniqueness of the infinite-volume Gibbs measure) for $q = 10$ on the triangular lattice. This improves an earlier result of Salas and Sokal [27] which used a machine-assisted proof to show strong spatial mixing for $q = 11$. Our approach can also be used to show (with computational assistance) strong spatial mixing for $q = 6$ on the rectangular lattice. This gives an alternative proof of the result of Achlioptas, Molloy, Moore, and Van Bussel [1] (which was also proved with machine assistance).

As we have previously mentioned, by using standard techniques our results can be used to show rapid mixing for Glauber dynamics for a wide class of graphs; see section 7 for full details.

2. Exponential decay and edge discrepancies. Let R be a nonempty finite region of the graph. For most of the technical part of this paper it will be convenient to consider the edge-boundary of R rather than the boundary ∂R of vertices surrounding R . Here is the notation that we will use. The *boundary* of the region R is the collection of all edges that have exactly one endpoint in R . A *coloring* of the boundary is a function from the set of edges in the boundary to the set $\{0\} \cup Q$.

Let R be a finite region and let B be a coloring of its boundary. A coloring C of R is said to be *proper* if

- adjacent vertices in R receive different colors, and
- vertices in R receive colors different from adjacent boundary edges.

Let $S(B)$ denote the set of proper colorings of R and let π_B be the uniform distribution on $S(B)$. We will be interested in studying how much $S(B)$ varies when we change the boundary coloring B by recoloring a single edge. This small change to the boundary is formalized in the following notation.

A *boundary pair* X consists of

- a nonempty finite region R_X of the graph,
- a distinguished edge s_X on the boundary of R_X , and
- a pair (B_X, B'_X) of colorings of the boundary of R_X which differ only on the edge s_X . We require that the two colors $B_X(s_X)$ and $B'_X(s_X)$ are both in Q . That is, the two boundary colorings differ on the coloring of edge s_X , but this edge is not an unconstrained edge (with color 0) in either boundary coloring.

For any boundary pair X , we define f_X to be the endpoint of s_X that is in R_X and w_X to be the other endpoint of s_X . Let E_X be the set of edges which connect f_X to another vertex in R_X . A *coupling* Ψ of π_{B_X} and $\pi_{B'_X}$ is a distribution on $S(B_X) \times S(B'_X)$ which has marginal distributions of π_{B_X} and $\pi_{B'_X}$. For such a coupling Ψ , we define $1_{\Psi, f}$ to be the indicator random variable for the event that, when a pair of colorings is drawn from Ψ , the color of f differs in these two colorings.

For any boundary pair X we define Ψ_X to be some coupling of π_{B_X} and $\pi_{B'_X}$ minimizing $\mathbb{E}[1_{\Psi, f_X}]$. For every pair of colors c and c' , let $p_X(c, c')$ be the probability that, when a pair of colorings (C, C') is drawn from Ψ_X , f_X is colored with color c in C and with color c' in C' .

Suppose that X is a boundary pair and that f is a vertex in R_X . Let $d(f, s_X)$ denote the distance within R_X from f to s_X . Thus, $d(f_X, s_X) = 1$, and if vertex f in R_X adjoins f_X , then $d(f, s_X) = 2$ and so on.

A main objective is to prove that the effect of the discrepancy at the boundary edge s_X decays exponentially with the distance from s_X (see Lemma 19). In order to do this, we use a recursive coupling (Lemma 12). The technique in Lemma 12 does not require that the graph be triangle-free—the general technique should also be applicable to models other than colorings.

To aid our analysis, we define a labelled tree T_X associated with each boundary pair X . The tree T_X is constructed as follows. Start with a vertex r which will be the root of T_X . For every pair of colors $c \in Q$ and $c' \in Q$, add an edge labelled $(p_X(c, c'), f_X)$ from r to a new node $r_{c,c'}$. If E_X is empty, $r_{c,c'}$ is a leaf. Otherwise, let e_1, \dots, e_k be the edges in E_X . For each $i \in \{1, \dots, k\}$, let $X_i(c, c')$ be the boundary pair consisting of

- the region $R_X - f_X$;
- the distinguished edge e_i ;
- the coloring B of the boundary of $R_X - f_X$ that
 - agrees with B_X on common edges,
 - colors e_1, \dots, e_{i-1} with color c' , and
 - colors e_i, \dots, e_k with color c ; and
- the coloring B' that agrees with B except that it colors e_i with color c' .

Recursively construct $T_{X_i(c,c')}$, the tree corresponding to boundary pair $X_i(c, c')$. Add an edge with label $(1, \cdot)$ from $r_{c,c'}$ to the root of $T_{X_i(c,c')}$. That completes the construction of T_X .

We say that an edge e of T_X is *degenerate* if the second component of its label is “.”. For edges e and e' of T_X , we write $e \rightarrow e'$ to denote the fact that e is an ancestor of e' . That is, either $e = e'$ or e is a proper ancestor of e' . Define the *level* of edge e to be the number of nondegenerate edges on the path from the root down to, and including, e . Suppose that e is an edge of T_X with label (p, f) . We say that the *weight* $w(e)$ of edge e is p . Also the *name* $n(e)$ of edge e is f . The *likelihood* $\ell(e)$ of e is $\prod_{e':e' \rightarrow e} w(e')$. The *cost* $\gamma(f, T_X)$ of a vertex f in T_X is $\sum_{e:n(e)=f} \ell(e)$.

LEMMA 12. *For every boundary pair X there exists a coupling Ψ of π_{B_X} and $\pi_{B'_X}$ such that, for all $f \in R_X$, $\mathbb{E}[1_{\Psi,f}] \leq \gamma(f, T_X)$.*

Proof. The coupling Ψ is constructed recursively in the same manner as the tree T_X , where at each stage the discrepancy at a given vertex is broken into discrepancies at single edges, so at every stage of the recursion we need only consider a pair of colorings with a discrepancy at a single edge (i.e., a boundary pair).

Let (C, C') denote the random variable corresponding to a pair of colorings from Ψ . If $|R_X| = 1$, then $\Psi = \Psi_X$. Otherwise, let e_1, \dots, e_k be the edges in E_X , i.e., those that are adjacent both to f_X and to another vertex in R_X . We will use Ψ_X to couple the coloring of vertex f_X and we will recursively construct a different coupled coloring of the other vertices in R_X . We will assign $C(f_X) = c$ and $C'(f_X) = c'$ with probability $p_X(c, c')$.

Let $X(c, c')$ be an “extended boundary pair” consisting of

- the region $R_X - f_X$;
- the coloring B_c of the boundary of $R_X - f_X$ that agrees with B_X on common edges and colors edges in E_X with color c ; and
- the coloring $B_{c'}$ of the boundary of $R_X - f_X$ that agrees with B_X on common edges and colors edges in E_X with color c' .

We can complete the coupling Ψ by constructing (for each c and c') a coupling $\Psi_{c,c'}$ of π_{B_c} and $\pi_{B_{c'}}$. (The reader may verify that this ensures that the marginal dis-

tributions of Ψ are correct.) The particular choice that we make for $\Psi_{c,c'}$ is either the perfect coupling if $c = c'$ or, if $c \neq c'$, the composition³ of $\Psi_1(c, c'), \dots, \Psi_k(c, c')$, where $\Psi_i(c, c')$ is a recursively-constructed coupling for boundary pair $X_i(c, c')$.

We will now show that, for all $f \in R_X$, $\mathbb{E}[1_{\Psi,f}] \leq \gamma(f, T_X)$. The proof is by induction on $|R_X|$. If $f = f_X$, then $\mathbb{E}[1_{\Psi,f}] = \gamma(f, T_X)$ by the construction of Ψ and T_X . This handles the base case, $|R_X| = 1$. Suppose $f \neq f_X$ and $|R_X| > 1$. Then

$$\begin{aligned} \mathbb{E}[1_{\Psi,f}] &= \sum_{c,c'} p_X(c, c') \mathbb{E}[1_{\Psi_{c,c'},f}] \leq \sum_{c,c'} p_X(c, c') \sum_{i=1}^k \mathbb{E}[1_{\Psi_i(c,c'),f}] \\ &\leq \sum_{c,c'} p_X(c, c') \sum_{i=1}^k \gamma(f, T_{X_i(c,c')}) = \gamma(f, T_X), \end{aligned}$$

where the second inequality uses the inductive hypothesis. \square

Suppose that X is a boundary pair. Lemma 12 ensures that there is a coupling of π_{B_X} and $\pi_{B'_X}$ with substantial agreement as long as, for most vertices $f \in R_X$, $\gamma(f, T_X)$ is small. A key ingredient from the construction of T_X which affects $\gamma(f, T_X)$ is the quantity $\mathbb{E}[1_{\Psi_X, f_X}]$, which we denote $\nu(X)$. (Thus, $\nu(X) = \min_{\Psi} \mathbb{E}[1_{\Psi, f_X}]$, where the minimum is over all couplings Ψ of π_{B_X} and $\pi_{B'_X}$.) An important part of our method is to determine good upper bounds on $\nu(X)$.

Let $d = B_X(s_X)$ and $d' = B'_X(s_X)$. Let $Q' = Q - \{d, d'\}$. For $c \neq d$, let $n_c(X)$ denote the number of colorings in $S(B_X)$ which color f_X with color c . Let $n_d(X)$ denote the number of colorings in $S(B'_X)$ which color f_X with color d . Let $N(X) = \sum_{c \in Q'} n_c(X)$. Define $\mu(X)$ to be the maximum of the probabilities $\Pr_{\pi_{B_X}}(f_X = d')$ and $\Pr_{\pi_{B'_X}}(f_X = d)$. That is,

$$\mu(X) = \frac{\max(n_d(X), n_{d'}(X))}{N(X) + \max(n_d(X), n_{d'}(X))}.$$

We use the following straightforward lemma to derive upper bounds on $\nu(X)$. Figure 1 is an illustration of part (ii) of this lemma. The basic idea is to pick a subregion R' that contains the vertex f_X . Compute the maximum value of μ for that subregion, where we maximize over colorings of the boundary of R' that agree with $B(R)$ on the common overlap of these boundaries. This maximum value is an upper bound for $\nu(X)$.

LEMMA 13. *Suppose that X is a boundary pair. Let R' be any subset of R_X which includes f_X . Let χ be the set of boundary pairs $X' = (R_{X'}, s_{X'}, B_{X'}, B'_{X'})$ such that $R_{X'} = R'$, $s_{X'} = s_X$, $B_{X'}$ agrees with B_X on common edges, and $B'_{X'}$ agrees with B'_X on common edges. Then $\nu(X) \leq \max_{X' \in \chi} \mu(X')$.*

Proof. We will show that

- (i) $\nu(X) \leq \mu(X)$ and
- (ii) $\mu(X) \leq \max_{X' \in \chi} \mu(X')$.

Let X be a boundary pair. To shorten the notation we will use n_c to denote $n_c(X)$ and N to denote $N(X)$. Let $d = B_X(s_X)$ and $d' = B'_X(s_X)$. Let $Q' = Q - \{d, d'\}$.

³The composition that we have in mind is the natural one—to choose a pair (σ_0, σ_k) from $\Psi_{c,c'}$ first choose (σ_0, σ_1) from $\Psi_1(c, c')$. Say $\sigma_0 = x_0$ and $\sigma_1 = x_1$. Then choose (σ_1, σ_2) from the conditional distribution of $\Psi_2(c, c')$, conditioned on $\sigma_1 = x_1$. Say that $\sigma_2 = x_2$. Now choose (σ_2, σ_3) from the conditional distribution of $\Psi_3(c, c')$ conditioned on $\sigma_2 = x_2$, and so on. This is the same as the composition that occurs in path coupling [4].

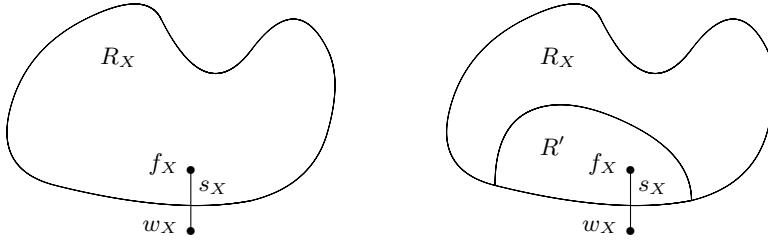


FIG. 1. Fix a subregion R' containing f_X . Then maximize $\mu(X')$ over boundary colorings of R' that agree with $B(R)$ on the overlapping part of the boundary.

For (i), we can construct a coupling Ψ of π_{B_x} and $\pi_{B'_x}$ which matches N colorings, each of which occurs with probability of at least

$$\frac{1}{\max(|S(B_X)|, |S(B'_X)|)} = \frac{1}{N + \max(n_d, n_{d'})}.$$

Thus,

$$\nu(X) \leq 1 - \frac{N}{N + \max(n_d, n_{d'})} = \mu(X).$$

(In fact, $\nu(X) = \mu(X)$, but we will not need this fact.)

Part (ii) will follow from the fact that $\Pr_{\pi_{B_X}}(f_X = d')$ is a convex combination of $\Pr_{\pi_{B_{X'}}}(f_X = d')$ for $X' \in \mathcal{X}$ and that $\Pr_{\pi_{B'_X}}(f_X = d)$ can be decomposed similarly. Let $W = R_X - R_{X'}$. Let H be the set of colorings of W . For $c \neq d$ and $\rho \in H$, let $n_{c,\rho}$ denote the number of colorings in $S(B_X)$ which color f_X with color c and W with coloring ρ . For $\rho \in H$, let $n_{d,\rho}$ denote the number of colorings in $S(B'_X)$ which color f_X with color d and W with coloring ρ . Let $N_\rho = \sum_{c \in Q'} n_{c,\rho}$. Then

$$\begin{aligned} \mu(X) &= 1 - \frac{N}{N + \max(n_d, n_{d'})} \\ &= 1 - \frac{\sum_{\rho \in H} N_\rho}{\sum_{\rho \in H} N_\rho + \max(\sum_{\rho \in H} n_{d,\rho}, \sum_{\rho \in H} n_{d',\rho})} \\ &\leq 1 - \frac{\sum_{\rho \in H} N_\rho}{\sum_{\rho \in H} N_\rho + \sum_{\rho \in H} \max(n_{d,\rho}, n_{d',\rho})} \\ &= \frac{\sum_{\rho \in H} \max(n_{d,\rho}, n_{d',\rho})}{\sum_{\rho \in H} (N_\rho + \max(n_{d,\rho}, n_{d',\rho}))} \\ &\leq \max_{\rho \in H} \frac{\max(n_{d,\rho}, n_{d',\rho})}{N_\rho + \max(n_{d,\rho}, n_{d',\rho})} \\ &= \max_{\rho \in H} \mu(X'), \end{aligned}$$

where X' is the boundary pair that is induced by ρ . \square

3. Bounding $\mu(X)$. In this section we show how to bound $\mu(X)$ for triangle-free graphs with sufficiently many colors. So that we can separate the task of bounding $\mu(X)$ from the task of showing strong spatial mixing, we define the notion of “ ε -good.” Informally, the number of colors q will be ε -good for a graph G (for some $\varepsilon \in (0, 1)$) whenever we can show strong spatial mixing for q -colorings of G .

DEFINITION 14. Suppose $\varepsilon \in (0, 1)$. We will say that the number of colors, q , is ε -good for the graph G if

$$\mu(X) \leq \frac{1}{\max(1, r)} \left(\frac{1}{1 + \varepsilon} \right)$$

for every boundary pair X in which R_X consists of a node f_X plus $r \geq 0$ neighbors y_1, \dots, y_r of f_X .

The purpose of this section is to prove Lemma 15 given below, which enables us to establish strong spatial mixing whenever $q > \alpha\Delta - \gamma$. The basic idea of the lemma is as follows. Consider a triangle-free region R_X and boundary condition \mathcal{B}_X . Suppose that the region contains sufficiently many neighbors of a vertex f_X , which is adjacent to the boundary. Then we derive an upper bound on the probability, in the equilibrium distribution, that f_X is assigned a particular color (in particular, the color d' from the definition of $\mu(X)$).

LEMMA 15. Let α be the solution to $\alpha^\alpha = e$ (so $\alpha \approx 1.76322$), and $\gamma = \frac{4\alpha^3 - 6\alpha^2 - 3\alpha + 4}{2(\alpha^2 - 1)} \approx 0.47031$. Suppose that the graph G is triangle-free and that for some $\Delta \geq 3$ the maximum degree of G is at most Δ . If $q \geq \alpha\Delta - \gamma + \alpha\varepsilon(\Delta - 1)$, then q is ε -good for G .

We prove Lemma 15 by reducing to the case in which R_X contains only f_X and its neighbors. We then use the fact that the graph has no triangles to count the number of colorings as a product. This leaves us with an optimization problem, the solution of which gives the result. Before we can prove Lemma 15, it helps to prove a preliminary lemma.

LEMMA 16. Let r, q, Δ be integers satisfying $q > \Delta > r \geq 2$. Define $p = q - \Delta + 1 \geq 2$. Consider a set of $\{0, 1\}$ -variables $\{\delta_{c,j} : 1 \leq c \leq q; 1 \leq j \leq r\}$ and an integer q' , subject to the bounds

$$s_j = \sum_{c=1}^q \delta_{c,j} \geq p \text{ for } 1 \leq j \leq r, \quad q \geq q' \geq p + r.$$

Define

$$n_c = \prod_{j=1}^r (s_j - \delta_{c,j}) \text{ for } 1 \leq c \leq q', \quad \text{and} \quad Z = \sum_{c=3}^{q'} n_c / n_1.$$

Then a minimal value of Z is attained by taking $q' = p + r$ and some choice of δ 's such that $s_j = p$ for all j and $\delta_{c,j} = 0$ if $c \notin \{3, \dots, q'\}$.

Proof. The choice $q' = p + r$ is clearly optimal. Fix some j and consider the dependence of Z on $\delta_{c,j}$ for all c , $1 \leq c \leq q'$. For some positive $a_1, \dots, a_{q'}$, we can write

$$Z = \sum_{c=3}^{q'} \frac{a_c(s_j - \delta_{c,j})}{a_1(s_j - \delta_{1,j})}.$$

We now suppose that Z is minimal and derive the properties claimed. First, we can ensure that $\delta_{c,j} = 0$ for $c \notin \{3, \dots, q'\}$. If any of these values $\delta_{c,j}$ is positive, we can set it to zero without increasing Z . If s_j had been at its lower bound of p , then we can restore this value by increasing $\delta_{c,j}$ from 0 to 1 for some $c \in \{3, \dots, q'\}$. Note that such a c exists since $q' - 2 \geq p$.

Now $n_1 = a_1 s_j$ (since $\delta_{1,j} = 0$), and

$$Z = \sum_{c=3}^{q'} \frac{a_c}{a_1} \left(1 - \frac{\delta_{c,j}}{s_j} \right) = A - \frac{1}{a_1} \frac{\sum_{c=3}^{q'} a_c \delta_{c,j}}{\sum_{c=3}^{q'} \delta_{c,j}}, \text{ where } A = \sum_{c=3}^{q'} \frac{a_c}{a_1}.$$

Since $\sum_{c=3}^{q'} a_c \delta_{c,j} / \sum_{c=3}^{q'} \delta_{c,j}$ is the average of s_j of the a 's, a maximal value for this quotient can be obtained by taking $s_j = p$, i.e., as small as possible, and selecting a set of p largest a 's with the corresponding δ 's. \square

Here is the proof of Lemma 15.

Proof. We will show that q is ε -good for G , assuming that $q \geq \alpha\Delta - \gamma + \varepsilon\alpha(\Delta - 1)$. Again, let $p = q - \Delta + 1$.

Suppose that X is a boundary pair in which R_X consists of a node f_X plus $r \leq 1$ neighbors. By Part (ii) of Lemma 13, $\mu(X) \leq \max_{X'} \mu(X')$, where X' is a boundary pair containing f_X only. The numerator of $\mu(X')$ is at most 1. The denominator is at least $q - \Delta$, so $\mu(X) \leq 1/(q - \Delta)$. Now we have $q - \Delta \geq (\alpha - 1)\Delta - \gamma + \varepsilon\alpha(\Delta - 1) > 1 + \varepsilon$ for $\Delta \geq 3$, so $\mu(X) < 1/(1 + \varepsilon)$.

For a boundary pair X we will use the notation $\mu_1(X)$ to denote $n_1(X)/(N(X) + n_1(X))$; define $\mu_2(X)$ similarly. We will show that, for every boundary pair X in which R_X consists of a node f_X plus $r > 1$ neighbors y_1, \dots, y_r of f_X , we have $\mu_1(X) \leq \frac{1}{r}(\frac{1}{1+\varepsilon})$. By symmetry, every such pair has the same upper bound on $\mu_2(X)$ and therefore on $\mu(X)$.

Suppose without loss of generality that $B_X(s_X) = 1$ and $B'_X(s_X) = 2$. Let K be the set of all colors which B_X assigns to neighbors of f_X other than s_X . We can assume without loss of generality that color 1 is not in K . Otherwise, $\mu_1(X) = 0$. Let $\delta_{c,j}$ be the Boolean indicator variable which is 0 if color c is used at a neighbor of y_j in the boundary of R_X . Let $Q' = Q - K$.

Now for every $c \in Q - Q'$ we have $n_c(X) = 0$. Since the graph is triangle-free, every $c \in Q'$ satisfies $n_c(X) = \prod_{j=1}^r (s_j - \delta_{c,j})$, where $s_j = \sum_{c=1}^q \delta_{c,j}$. Thus,

$$\frac{1}{\mu_1(X)} - 1 = \frac{\sum_{c=3}^q n_c(X)}{n_1(X)} = \frac{\sum_{c \in Q' - \{1,2\}} n_c(X)}{n_1(X)}.$$

Without loss of generality, we can assume that the colors in $Q' - \{1, 2\}$ are colors $3, \dots, q'$, for some q' , so we have

$$(1) \quad \frac{1}{\mu_1(X)} - 1 = \frac{\sum_{c=3}^{q'} n_c(X)}{n_1(X)}.$$

We are now in the framework of Lemma 16. We have the constraints $s_j \geq p$ since the degree of y_j is at most Δ , and $q' \geq p + r$ since $q' - 2 = |Q' - \{1, 2\}| \geq q - 2 - (\Delta - (r + 1))$. Let Z be the right-hand side of (1). Z is minimized by taking $q' = p + r$ and some choice of δ 's such that $s_j = p$ for all j and $\delta_{c,j} = 0$ if $c \notin \{3, \dots, q'\}$. Writing $m_c = \sum_{j=1}^r \delta_{c,j}$ and plugging in $n_c(X) = \prod_{j=1}^r (s_j - \delta_{c,j})$, we have

$$(2) \quad \begin{aligned} \frac{1}{\mu_1(X)} - 1 = Z &\geq \sum_{c=3}^{q'} \prod_{j=1}^r \left(1 - \frac{\delta_{c,j}}{p} \right) \\ &= \sum_{c=3}^{q'} \left(1 - \frac{1}{p} \right)^{m_c}, \end{aligned}$$

where $\sum_{c=3}^{q'} m_c = \sum_{j=1}^r s_j = rp$, and $q' - 2 = p + r - 2 \geq p$.

Our goal is to derive a lower bound for $(Z + 1)/r$ with respect to r and the m_c 's. Let S denote the expression (2) for an arbitrary choice of m_c 's, and consider the effect of increasing r by 1. The sum of the m_c 's is increased by p and the number of them, q' , is increased by 1. One possibility is to leave the existing m_c 's unchanged and add the extra term $(1 - 1/p)^p$, corresponding to the new m_c . To show that the lower bound given by minimizing $(S + 1)/r$ with respect to the m_c 's is decreasing in r , it is sufficient to verify that

$$\frac{S + 1}{r} \geq \frac{S + 1 + \left(1 - \frac{1}{p}\right)^p}{r + 1}, \text{ i.e., } S + 1 \geq r \left(1 - \frac{1}{p}\right)^p.$$

The expression S is minimized by taking the m_c 's as equal as possible, so

$$S \geq (q' - 2) \left(1 - \frac{1}{p}\right)^{\frac{rp}{q' - 2}} \geq r \left(1 - \frac{1}{p}\right)^p,$$

since $q' - 2 \geq r$. Thus the smallest lower bound that we get is derived from the expression S with r taking its maximum value of $\Delta - 1$ (so $q' = q$) and the m_c 's being taken as nearly equal as possible subject to integrality constraints. The same bound holds for other choices of r and the m_c 's.

We therefore define

$$J(q, \Delta) = (q - 2 - v) \left(1 - \frac{1}{p}\right)^u + v \left(1 - \frac{1}{p}\right)^{u+1} = (q - 2) \left(1 - \frac{1}{p}\right)^u \left(1 - \frac{v}{(q - 2)p}\right),$$

where $u = \lfloor (\Delta - 1)p/(q - 2) \rfloor$ and $v = (\Delta - 1)p \bmod (q - 2)$, and also define

$$J'(q, \Delta) = (q - 2) \left(1 - \frac{1}{p}\right)^{\frac{p(\Delta - 1)}{q - 2}}.$$

Note that $J \geq J'$, since J and J' are minimizations with and without the constraint of integral m_c 's, respectively. We have $Z \geq J(q, \Delta) \geq J'(q, \Delta)$. To prove that q is ε -good we need to show that $Z/(\Delta - 1) \geq 1 + \varepsilon - 1/(\Delta - 1)$. For the current lemma we just use the simpler expression J' in the proof. (We observe later that by using the inequality based on J we may obtain a slight improvement for the lower bound on q for some values of Δ ; see the remark following Theorem 5 on page 18.)

Define $x = 1 - (q - 2)/(\alpha(\Delta - 1))$, so that

$$\frac{J'(q, \Delta)}{\Delta - 1} = \alpha(1 - x) \left(1 - \frac{1}{p}\right)^{\frac{p}{\alpha(1 - x)}}.$$

We use two simple inequalities.

LEMMA 17.

- (i) $-\ln(1 - x) \leq \frac{x}{1 - x}$ for $-\infty < x < 1$.
- (ii) $-p \ln\left(1 - \frac{1}{p}\right) < 1 + \frac{1}{2(p - 1)}$ for $p > 1$.

Proof. For (i), let $f(x) = x/(1 - x) + \ln(1 - x)$, so that $f(0) = 0$. Since $df/dx = x/(1 - x)^2$ has the sign of x , the inequality holds. For (ii), it is enough to compare the power series expansions in $1/p$. \square

Note that

$$(3) \quad p - 1 = q - \Delta > \alpha\Delta - \gamma - \Delta > (\alpha - 1)(\Delta - 1).$$

Applying Lemma 17, the equality $\alpha \ln \alpha = 1$, and inequality (3), we then derive

$$\begin{aligned} \ln \left(\frac{J'(q, \Delta)}{\Delta - 1} \right) &= \ln \alpha + \ln(1 - x) + \frac{p}{\alpha(1 - x)} \ln \left(1 - \frac{1}{p} \right) > \ln \alpha - \frac{x}{1 - x} - \frac{1 + \frac{1}{2(p-1)}}{\alpha(1 - x)} \\ &= \frac{\alpha \ln \alpha (1 - x) - \alpha x - 1 - \frac{1}{2(p-1)}}{\alpha(1 - x)} = \frac{-(\alpha + 1)x - \frac{1}{2(p-1)}}{\alpha(1 - x)} \\ (4) \quad &> \frac{-(\alpha + 1)x - \frac{1}{2(\alpha-1)(\Delta-1)}}{\alpha(1 - x)} = \frac{\alpha + 1}{\alpha} - \frac{\alpha + 1 + \frac{1}{2(\alpha-1)(\Delta-1)}}{\alpha(1 - x)}. \end{aligned}$$

Since

$$1 - x = \frac{q - 2}{\alpha(\Delta - 1)} = \frac{q - \alpha\Delta + \gamma}{\alpha(\Delta - 1)} + \frac{\alpha(\Delta - 1) - (2 + \gamma - \alpha)}{\alpha(\Delta - 1)} \geq \varepsilon + 1 - \frac{2 + \gamma - \alpha}{\alpha(\Delta - 1)},$$

we may write $w = 1/(\Delta - 1)$ and give the following lower bound for the right-hand side of (4):

$$\frac{\alpha + 1}{\alpha} - \frac{\frac{\alpha+1}{\alpha} + \frac{w}{2\alpha(\alpha-1)}}{1 + \varepsilon - \frac{(2+\gamma-\alpha)w}{\alpha}} = A - \frac{B(\varepsilon)}{C(\varepsilon) - Dw} = F(\varepsilon, w), \text{ say,}$$

where

$$B(\varepsilon) = \frac{\alpha + 1}{\alpha} + \frac{1 + \varepsilon}{2(\alpha - 1)(2 + \gamma - \alpha)} > 0, \quad A = B(0), \quad C(\varepsilon) = 1 + \varepsilon, \quad \text{and} \quad D = \frac{2 + \gamma - \alpha}{\alpha}.$$

Our remaining goal in proving that q is ε -good is to show that

$$(5) \quad F(\varepsilon, w) \geq \ln(1 + \varepsilon - w), \text{ for } 0 \leq \varepsilon \leq 1, \text{ and } 0 < w \leq 1/2.$$

We first show that

$$\frac{\partial(F - \ln(1 + \varepsilon - w))}{\partial w} = -\frac{DB(\varepsilon)}{(C(\varepsilon) - Dw)^2} + \frac{1}{1 + \varepsilon - w} > 0.$$

Since $1 + \varepsilon - w > 0$, this is equivalent to checking that $(C(\varepsilon) - Dw)^2 - (1 + \varepsilon - w)DB(\varepsilon) > 0$. Numerically, this polynomial in ε and w is approximately

$$0.6285\varepsilon^2 + (0.6285 - 0.4305w)\varepsilon + 0.1980w + 0.1608w^2,$$

which is clearly positive⁴ since $w \leq 0.5$. The constant term

$$C(0)^2 - DB(0) = \frac{4\alpha^3 - 6\alpha^2 - 3\alpha + 4 - 2\gamma(\alpha^2 - 1)}{2\alpha^2(\alpha - 1)}$$

is zero by the choice of γ . It is sufficient therefore to verify the inequality (5) for $w = 0$.

⁴To verify formally that the expression is positive, one can derive upper and lower bounds on the coefficients using upper and lower bounds on α .

To show that $F(\varepsilon, 0) - \ln(1 + \varepsilon) \geq 0$, we first verify that the second derivative is negative, and then merely check the inequality at the extreme values, $\varepsilon = 0, 1$.

However,

$$\begin{aligned} \frac{d^2}{d\varepsilon^2}(F(\varepsilon, 0) - \ln(1 + \varepsilon)) &= \frac{d^2}{d\varepsilon^2} \left(\frac{\alpha + 1}{\alpha} \left(1 - \frac{1}{1 + \varepsilon} \right) - \ln(1 + \varepsilon) \right) \\ &= -\frac{\alpha + 1}{\alpha} \frac{2}{(1 + \varepsilon)^3} + \frac{1}{(1 + \varepsilon)^2} = \frac{-2(\alpha + 1) + \alpha(1 + \varepsilon)}{\alpha(1 + \varepsilon)^3} < 0, \end{aligned}$$

$$F(0, 0) - \ln 1 = 0, \text{ and } F(1, 0) - \ln(1 + 1) = \frac{\alpha + 1}{2\alpha} - \ln 2 \approx 0.0904 > 0.$$

This completes the verification. \square

We now show how to use Lemma 15 to prove that the effect of a discrepancy at the boundary edge s_X decays exponentially with the distance from s_X .

Let X be a boundary pair. For any $d \geq 1$, let $E_d(X)$ denote the set of level- d edges in T_X . Let $\Gamma_d(X) = \sum_{e \in E_d(X)} \ell(e)$. In Lemma 18 below we show that $\Gamma_d(X)$ is exponentially small in d . Say that a boundary pair X is in \mathcal{N}_i (for $i \in \{0, \dots, \Delta - 1\}$) if exactly i of the neighbors of f_X are in R_X . Let Γ_d be the maximum of $\Gamma_d(X)$, maximized over all boundary pairs X .

LEMMA 18. *Suppose that q is ε -good for G . Then for every boundary pair X and any $d \geq 1$, $\Gamma_d(X) \leq (1 + \varepsilon)^{-d}$.*

Proof. The proof is by induction on d . For the base case, $d = 1$, note that for any boundary pair X , $\Gamma_1(X) \leq \nu(X)$. Now apply Lemma 13 with $R' = \{f_X\}$, and by the given upper bound on $\mu(X)$ (and the definition of ε -good), we find that $\nu(X) \leq 1/(1 + \varepsilon)$. For the inductive step, suppose that $X \in \mathcal{N}_r$. Then using the definition of ε -good again (and Lemma 13), we see that $\Gamma_d(X) \leq \nu(X) \cdot r \cdot \Gamma_{d-1}$. \square

LEMMA 19. *Suppose that q is ε -good for G . Then for every boundary pair X there exists a coupling Ψ of π_{B_X} and $\pi_{B'_X}$ such that, for all $f \in R_X$,*

$$\mathbb{E}[1_{\Psi, f}] \leq \frac{1}{\varepsilon} (1 + \varepsilon)^{-d(f, s_X) + 1}.$$

Furthermore,

$$\sum_{f \in R_X} \mathbb{E}[1_{\Psi, f}] \leq \frac{1}{\varepsilon}.$$

Proof. By Lemma 12, $\mathbb{E}[1_{\Psi, f}] \leq \gamma(f, T_X)$. Furthermore, $\gamma(f, T_X) = \sum_{e: n(e)=f} \ell(e)$, which is at most $\sum_{d \geq d(f, s_X)} \Gamma_d(X)$. By Lemma 18, this is at most

$$\sum_{d \geq d(f, s_X)} (1 + \varepsilon)^{-d} = (1 + \varepsilon)^{-d(f, s_X)} \sum_{d \geq 0} (1 + \varepsilon)^{-d} = (1 + \varepsilon)^{-d(f, s_X)} \frac{1 + \varepsilon}{\varepsilon}.$$

Similarly,

$$\sum_{f \in R_X} \mathbb{E}[1_{\Psi, f}] \leq \sum_{f \in R_X} \gamma(f, T_X) = \sum_{f \in R_X} \sum_{e: n(e)=f} \ell(e) = \sum_{e: n(e) \in R_X} \ell(e) = \sum_{d \geq 1} \Gamma_d(X). \quad \square$$

4. Exponential decay, vertex discrepancies, and strong spatial mixing.

Lemma 19 shows that the effect of a discrepancy at a boundary edge decays exponentially with the distance from that edge. In this section we show that the same holds for a discrepancy at a boundary vertex. This enables us to show that the collection of finite-volume Gibbs measures corresponding to the uniform distribution on proper colorings has strong spatial mixing.

A *vertex-boundary pair* X consists of

- a nonempty finite region R_X of the graph;
- a distinguished vertex v_X in ∂R_X ; and
- a pair $(\mathcal{B}_X, \mathcal{B}'_X)$ of colorings of ∂R which differs only on vertex v_X . We require that the two colors $\mathcal{B}_X(v_X)$ and $\mathcal{B}'_X(v_X)$ are both in Q . That is, the two boundary colorings differ on the color of vertex v_X , but this vertex is not an unconstrained vertex (with color 0) in either boundary coloring.

Let $d(f, v_X)$ be the distance within R_X from a vertex f to vertex v_X .

LEMMA 20. *Suppose that q is ε -good for a graph G with degree at most Δ . For every vertex-boundary pair X there is a coupling Ψ of $\pi_{\mathcal{B}_X}$ and $\pi_{\mathcal{B}'_X}$ such that, for all $f \in R$,*

$$\mathbb{E}[1_{\Psi, f}] \leq \Delta \frac{1}{\varepsilon} (1 + \varepsilon)^{-d(f, v_X) + 1}.$$

Furthermore,

$$\sum_{f \in R_X} \mathbb{E}[1_{\Psi, f}] \leq \frac{\Delta}{\varepsilon}.$$

Proof. This follows from Lemma 19 using a union bound by breaking the difference in a single vertex into the sum of differences in the edges that bound it. Let e_1, \dots, e_k be the boundary edges of R_X that are adjacent to v_X . Let X_i be the boundary pair consisting of the region R_X , the distinguished edge e_i , a coloring B of the boundary of R_X that agrees with \mathcal{B}_X except that edges e_1, \dots, e_{i-1} are colored with color $\mathcal{B}'_X(v_X)$ and e_i, \dots, e_k are colored with color $\mathcal{B}_X(v_X)$, and a coloring B' that is the same as B except that it colors e_i with color $\mathcal{B}'_X(v_X)$. We construct a coupling of $\pi_{\mathcal{B}_X}$ and $\pi_{\mathcal{B}'_X}$ by composing couplings Ψ_1, \dots, Ψ_k of X_1, \dots, X_k . Now

$$\mathbb{E}[1_{\Psi, f}] \leq \sum_{i=1}^k \mathbb{E}[1_{\Psi_i, f}]. \quad \square$$

COROLLARY 21. *Suppose that q is ε -good for the graph G , and that the maximum degree, Δ , of G is bounded. Then the system specified by uniform finite-volume Gibbs measures on proper q -colorings of G has strong spatial mixing.*

Proof. Using Definition 1, we wish to show that there are constants β and $\beta' > 0$ such that for any vertex-boundary pair X and any $\Lambda \subseteq R_X$,

$$d_{\text{tv}}(\pi_{\mathcal{B}_X, \Lambda}, \pi_{\mathcal{B}'_X, \Lambda}) \leq \beta |\Lambda| \exp(-\beta' d(v_X, \Lambda)).$$

The total variation distance of $\pi_{\mathcal{B}_X, \Lambda}$ and $\pi_{\mathcal{B}'_X, \Lambda}$ is at most the probability that the induced colorings differ in the coupling Ψ from Lemma 20. This is at most $\sum_{f \in \Lambda} \mathbb{E}[1_{\Psi, f}]$. As in the proof of Lemma 20, Ψ is the composition of Ψ_1, \dots, Ψ_k . Following the proof of Lemma 19,

$$\sum_{f \in \Lambda} \mathbb{E}[1_{\Psi_i, f}] \leq \sum_{e: n(e) \in \Lambda} \ell(e) \leq \sum_{d \geq d(v_X, \Lambda)} \Gamma_d(X) \leq \frac{1}{\varepsilon} (1 + \varepsilon)^{-d(v_X, \Lambda) + 1}$$

so the total variation distance is at most

$$\frac{\Delta}{\varepsilon}(1 + \varepsilon)^{-d(v_X, \Lambda)+1}.$$

Now we can take $\beta = \frac{\Delta(1+\varepsilon)}{\varepsilon}$ and $\beta' = \log(1 + \varepsilon)$. \square

Remark. Note that the $|\Lambda|$ factor is not crucial in the definition of strong spatial mixing. In particular, our upper bound on the total variation distance does not use this factor.

Combining Corollary 21 with Lemma 15 we get the following result.

THEOREM 5. *Let α be the solution to $\alpha^\alpha = e$ (so $\alpha \approx 1.76322$), and $\gamma = \frac{4\alpha^3 - 6\alpha^2 - 3\alpha + 4}{2(\alpha^2 - 1)} \approx 0.47031$. Let G denote an infinite triangle-free graph, and suppose that for some $\Delta \geq 3$ the maximum degree of G is at most Δ . The spin system specified by uniform finite-volume Gibbs measures on proper q -colorings of G has strong spatial mixing if $q > \alpha\Delta - \gamma$.*

Remark. In the proof of Lemma 15 we show that the simple bound of $q \geq \alpha\Delta - \gamma + \alpha\varepsilon(\Delta - 1)$ is sufficient to guarantee that $(J'(q, \Delta) + 1)/(\Delta - 1) \geq 1 + \varepsilon$, which yields ε -goodness since $1/\mu_1(X) - 1 = Z \geq J(q, \Delta) \geq J'(q, \Delta)$. However, for any particular value of Δ we may use the bounds $(J(q, \Delta) + 1)/(\Delta - 1) - 1 \geq (J'(q, \Delta) + 1)/(\Delta - 1) - 1 \geq \varepsilon$ directly.

For example, with $\Delta = 4$ and $q = 7$, Lemma 15 gives ε -goodness for $\varepsilon \approx 0.079$, whereas the direct use of J' or J give values of about 0.169 or 0.177, respectively.

Of a little more interest are the rather sparse cases where the direct inequalities give a reduced value of q . Strong spatial mixing can be shown for $q = \lfloor \alpha\Delta - \gamma \rfloor$ for $\Delta = 19, 36, 74, 357, 2380, 148264, 686821, \dots$. In the two cases, $\Delta = 19, 74$, the bounds of $q = 33, 130$, respectively, require the use of J (with its integral constraints) rather than just J' .

5. Using the geometry of the lattice. In this section we consider the lattice \mathbb{Z}^3 . This is a triangle-free graph with degree 6, so Theorem 5 gives strong spatial mixing for $q \geq 11$. We will exploit the geometry of the lattice to show strong spatial mixing for $q = 10$. The idea is to use the geometry to derive a system of recurrences and to use these recurrences to construct the coupling.

We start by recording some upper bounds on $\mu(X)$. Let $\mu' = 125/589$ and let $\mu'' = 625/3121$. The following corollary follows from the proof of Lemma 15.

COROLLARY 22. *Suppose X is a boundary pair in which R_X consists of a node f_X plus $r \geq 0$ neighbors y_1, \dots, y_r of f_X . If $r = 0$, then $\mu(X) \leq 1/4$. If $r = 4$, then $\mu(X) \leq \mu'$. If $r = 5$, then $\mu(X) \leq \mu''$.*

Proof. The $r = 0$ case follows from the fact that $\mu(X) \leq 1/(q - \Delta) = 1/4$. For the other cases, we use the same reasoning that we used in the proof of Lemma 15 to determine $J(q, \Delta)$. Let $q' = q - \Delta + 1 + r$, $u = \lfloor \frac{rp}{q' - 2} \rfloor$ and $v = rp \bmod (q' - 2)$. (These are the same as the definitions in the proof of Lemma 15 except that there we specialized to $r = \Delta - 1$ so we had $q' = q$.) Let $h(q, \Delta, r)$ be the sum of the $q' - 2$ terms in (2) when we minimize by making the m_c 's as equal as possible. Namely,

$$h(q, \Delta, r) = (q' - 2 - v) \left(1 - \frac{1}{p}\right)^u + v \left(1 - \frac{1}{p}\right)^{u+1}.$$

It follows from (2) and from the argument in the proof of Lemma 15 that

$$\mu(X) \leq \frac{1}{1 + h(q, \Delta, r)}.$$

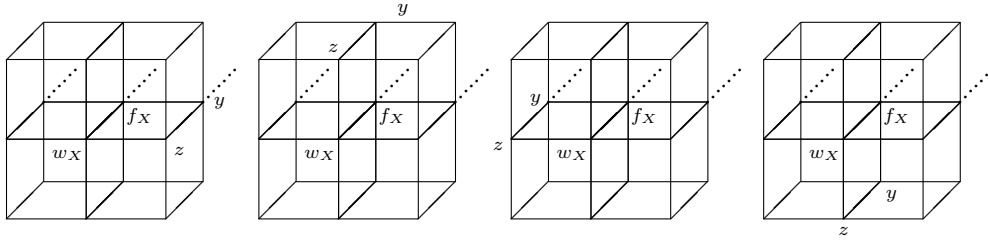


FIG. 2. Configurations in U with $y \notin R_X$, or those in V with $z \notin R_X$.

The values can then be calculated directly. \square

We would like to use the bounds in Corollary 22 to prove that $\Gamma_d(X)$ is exponentially small in d . The proof in Lemma 18 uses the following simple recursive idea. If $X \in \mathcal{N}_r$, then $\Gamma_d(X) \leq \nu(X) \cdot r \cdot \Gamma_{d-1}$. This idea suffices if our upper bound on $\nu(X)$ is less than $1/r$. This is not the case for the bounds in Corollary 22. However, it is a bit pessimistic to assume that all of the r recursive subproblems correspond to the worst recursive case Γ_{d-1} . Using the geometry of the lattice, we can keep track of the recursion and do better.

We start by defining some sets of boundary pairs.

We will say that a boundary pair is in the set U if either of the following conditions hold:

- Either $X \in \mathcal{N}_1 \cup \mathcal{N}_2 \cup \mathcal{N}_3$; or
- $X \in \mathcal{N}_4$ and the following is true. Let y be the neighbor of f_X that is not in R_X and is not equal to w_X . We require that the vertices w_X and y differ in exactly two coordinates (in 3-dimensional space); see Figure 2.

We will say that a boundary pair is in the set V if the following condition holds: There is a vertex $z \notin R_X$ and a vertex $y \neq w_X$ such that $z \sim w_X$ (meaning z is adjacent to w_X) and $z \sim y \sim f_X$; see Figure 2 again for the relevant configurations. Note that the subsets U and V of boundary pairs depend only on R_X and s_X (they do not depend on B_X or B'_X). Let $U_d = \max_{X \in U} \Gamma_d(X)$ and $V_d = \max_{X \in V} \Gamma_d(X)$. The next lemma follows from the geometry of the lattice.

LEMMA 23. *Suppose that $q = 10$ and G is \mathbb{Z}^3 . Suppose $d > 1$. Then*

$$\begin{aligned} \Gamma_d &\leq \max \left(4\mu' \Gamma_{d-1}, \mu'' (\Gamma_{d-1} + 4V_{d-1}), \frac{3}{4} \Gamma_{d-1} \right), \\ U_d &\leq \max \left(\mu' (\Gamma_{d-1} + 3V_{d-1}), \frac{3}{4} \Gamma_{d-1} \right), \\ V_d &\leq \max \left(4\mu' \Gamma_{d-1}, \mu'' (U_{d-1} + 4\Gamma_{d-1}), \frac{3}{4} \Gamma_{d-1} \right). \end{aligned}$$

Proof. Consider a boundary pair X . If $X \in \mathcal{N}_1 \cup \mathcal{N}_2 \cup \mathcal{N}_3$, then $\Gamma_d(X) \leq \nu(X) \cdot 3 \cdot \Gamma_{d-1}$. By Lemma 13 and Corollary 22 (with $r = 0$), $\nu(X) \leq 1/4$, so $\Gamma_d(X) \leq \frac{3}{4} \Gamma_{d-1}$ and one of the inequalities on Γ_d is satisfied. If $X \in \mathcal{N}_4$, then applying Corollary 22 with $r = 4$ we get $\Gamma_d(X) \leq \mu' 4\Gamma_{d-1}$, and the upper bound on Γ_d is satisfied. Otherwise, X is in \mathcal{N}_5 . Using the definition of V , we can deduce that

$$(6) \quad \Gamma_d(X) \leq \nu(X) (\Gamma_{d-1} + 4V_{d-1}),$$

which gives us the other bound on Γ_d (using Corollary 22 with $r = 5$). To see that inequality (6) is satisfied, let y be any of the 4 neighbors of f_X in R_X such that y and w_X differ on two coordinates. Now consider the recursive problem in which f_X

becomes the new w_X and y becomes the new f_X . The original w_X becomes the (new) z in the definition of V . So this recursive problem is in V .

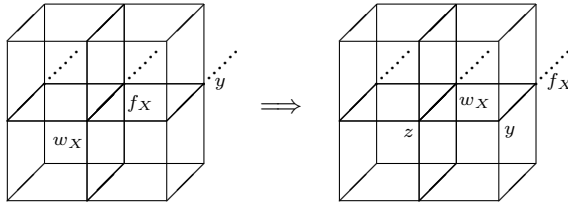


FIG. 3. The basic recursion step for a configuration.

Next, consider a boundary pair $X \in U$. We wish to show that the inequality on U_d is satisfied. This is straightforward if $X \in \mathcal{N}_1 \cup \mathcal{N}_2 \cup \mathcal{N}_3$, so suppose that X is in \mathcal{N}_4 . Let y denote the vertex in the definition of U . (So y is not in R_X and y and w_X differ in exactly two coordinates.) Using the definition of V , we can deduce that

$$\Gamma_d(X) \leq \nu(X)(\Gamma_{d-1} + 3V_{d-1}),$$

giving one of the upper bounds on U_d . (To see this, let y' be one of the three neighbors of f_X in R_X that differs from y in two coordinates. Note that the subproblem moving from f_X to y' is in V ; see Figure 4.)

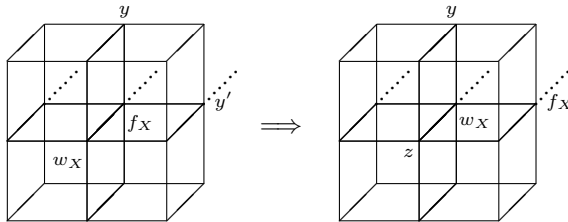


FIG. 4. The recursion step for a configuration in U .

Finally, consider a boundary pair $X \in V$. The inequality for V_d is similar to that for Γ_d except when $X \in \mathcal{N}_5$. In this case, let z and y be the vertices in the definition of V . Now note that the subproblem moving from f_X to y is in U . This gives the third equation. \square

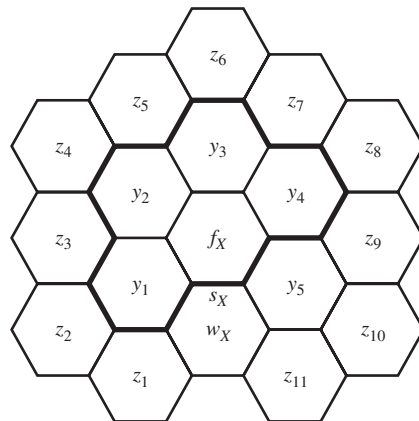
Here is the analogy of Lemma 18.

LEMMA 24. Let $\zeta = 0.0001$. Suppose that $q = 10$ and G is \mathbb{Z}^3 . For every boundary pair X and every $d \geq 1$, $\Gamma_d(X) \leq (1 - \zeta)^d$.

Proof. Let $u = 0.8294$ and $v = 0.968$. We will prove by induction on d that $\Gamma_d \leq (1 - \zeta)^d$ and $U_d \leq u(1 - \zeta)^d$ and $V_d \leq v(1 - \zeta)^d$. For the base case, $d = 1$, note that for any relevant boundary pair X , $\Gamma_1(X) \leq \nu(X)$. Now apply Lemma 13 with $R' = \{f_X\}$, and by Corollary 22 (with $r = 0$), we find that $\nu(X) \leq 1/4$. Thus, $\Gamma_1(x) \leq \min\{u, v, 1\}(1 - \zeta)$.

The inductive step follows directly from Lemma 23 since the following inequalities hold: $4\mu' \leq (1 - \zeta) \min(1, v)$, $\frac{3}{4} \leq (1 - \zeta) \min(1, v, u)$, $\mu''(1 + 4v) \leq (1 - \zeta)$, $\mu'(1 + 3v) \leq (1 - \zeta)u$, and $\mu''(u + 4) \leq (1 - \zeta)v$. \square

The proof of Theorem 6 now follows the argument in sections 3 and 4. The only difference is that instead of applying Lemma 18, we use Lemma 24 (which gives exactly the same result with $\varepsilon = \zeta/(1 - \zeta)$). The analogue of Lemma 20 provides an ε -coupling cover. Since \mathbb{Z}^3 is neighborhood-amenable, we obtain Corollary 10.

FIG. 5. *The triangular lattice.*

6. Computational assistance. In section 5 we showed strong spatial mixing for $q = 10$ and the degree-6 lattice \mathbb{Z}^3 . The same argument does not apply to the triangular lattice because we cannot apply Lemma 15 (or its proof) to a graph with triangles. Nevertheless, we can use our method with computational assistance to prove strong spatial mixing for the triangular lattice.

6.1. The lattice. A piece of the triangular lattice is depicted in Figure 5. Each vertex of the lattice is depicted as a hexagonal face in the picture. Every vertex has degree 6. Thus, the 6 neighbors of vertex f_X are w_X , y_1 , y_2 , y_3 , y_4 , and y_5 .

6.2. Relevant boundary pairs. In order to prove strong spatial mixing, we will need upper bounds on $\mu(X)$ similar to those obtained in Corollary 22. Since we will use computation, we want to restrict the search space as far as possible. We do that by defining the notion of a “relevant” boundary pair. Intuitively, the idea is that these boundary pairs are the ones that are induced by a pair of colorings of the vertex-boundary ∂R_X .

We say that a boundary pair is *relevant* if it is the case that any two *adjacent* edges on the boundary that share a vertex $f \notin R_X$ have the same color in at least one of the two colorings B_X and B'_X (and so in both of B_X and B'_X except when edge s_X is involved). For example, in Figure 5, if R_X consists of the five vertices enclosed by the thicker line (namely f_X , y_1 , y_2 , y_3 , and y_4) and X is a relevant boundary pair, then B_X and B'_X assign the same color to the edges (y_1, z_3) and (y_2, z_3) . Note that our definition of “relevant” is specific to the geometry of the lattice. The edges (y_1, z_2) and (y_1, z_3) are *adjacent* but the edges (y_1, z_3) and (y_1, z_1) are not adjacent.

It is important to observe that our recursive construction preserves “relevance.” That is, if X is a relevant boundary pair, then all of the boundary pairs in the tree T_X are also relevant. We can ensure this by refining the construction of T_X (see section 2) as follows. When the edges in E_X are given the names e_1, \dots, e_k , order these edges clockwise around the vertex f_X starting from s_X . This ordering ensures that e_i is not “adjacent” to e_j unless i and j differ by 1. Now note that if X is relevant, then so is the constructed boundary pair $X_i(c, c')$.

Next, note that Lemma 13 can be extended as follows. If the boundary pair X is relevant, then the boundary pair X' constructed in the proof is also relevant. Therefore, the set χ can be restricted to relevant boundary pairs. For convenience, we state

the extended lemma here.

LEMMA 25. *Suppose that X is a relevant boundary pair. Let R' be any subset of R_X which includes f_X . Let χ be the set of relevant boundary pairs $X' = (R_{X'}, s_{X'}, B_{X'}, B'_{X'})$ such that $R_{X'} = R'$, $s_{X'} = s_X$, $B_{X'}$ agrees with B_X on common edges, and $B'_{X'}$ agrees with B'_X on common edges. Then $\nu(X) \leq \max_{X' \in \chi} \mu(X')$.*

6.3. Bounding $\mu(X)$. By analogy to Corollary 22 we will now provide some upper bounds on $\mu(X)$ for the triangular lattice. Let $\mu' = 31/136$ and $\mu'' = 1111/4966$. We now give four lemmas bounding $\mu(X)$ for particular boundary pairs X .

LEMMA 26. *Suppose X is a boundary pair with $|R_X| = 1$. Then $\mu(X) \leq 1/4$.*

Proof. The numerator in the definition of $\mu(X)$ is at most 1. The denominator is at least $q - \Delta = 4$. \square

LEMMA 27. *Suppose X is a boundary pair in which R_X consists of f_X and one neighbor y of f_X . Then $\mu(X) \leq 5/21$.*

Proof. Let E_1 be the set of edges of f_X except for s_X and the edge between f_X and y . Let E_2 be the set of edges of y except for the edge between f_X and y . Let U be the set of colors that B_X (and so also B'_X) assigns to edges in E_1 , and similarly let V be the set of colors assigned to E_2 . Let $d = B_X(s_X)$ and $d' = B'_X(s_X)$. Let $Q' = Q - \{d, d'\}$. To shorten the notation we will use n_c to denote $n_c(X)$ and N to denote $N(X)$. Let $q_1 = |Q' \setminus U|$ and let $q_2 = |Q \setminus V|$. We see that $q_1 \geq 8 - |U| \geq 4$ and $q_2 \geq 10 - |V| \geq 5$.

For $c \in Q$ we see that

$$\begin{aligned} n_c &= 0 && \text{if } c \in U, \\ &= q_2 && \text{if } c \notin U \text{ and } c \in V, \\ &= q_2 - 1 && \text{if } c \notin U \text{ and } c \notin V. \end{aligned}$$

Thence, $\max(n_d, n_{d'}) \leq q_2$ and

$$N = \sum_{c \in Q'} n_c \geq (q_2 - 1)|Q' \setminus U| = (q_2 - 1)q_1.$$

Since $\mu(X)$ is monotone increasing in $\max(n_d, n_{d'})$ and decreasing in N , we have

$$\mu(X) \leq \frac{q_2}{q_2 + (q_2 - 1)q_1} = \frac{1}{1 + (1 - 1/q_2)q_1} \leq \frac{1}{1 + (1 - 1/5)4} = \frac{5}{21}. \quad \square$$

LEMMA 28. *Suppose X is a relevant boundary pair in which $R_X = \{f_X, y_1, y_2, y_3\}$ and f_X is adjacent to each of the y_i 's and $w_X \sim y_1 \sim y_2 \sim y_3$. Then $\mu(X) \leq \mu'$.*

Proof. By computation, we considered every such relevant boundary pair X (approximately 2×10^6 of them) and calculated $\mu(X)$. \square

LEMMA 29. *Suppose X is a relevant boundary pair in which $R_X = \{f_X, y_1, y_2, y_3, y_4\}$ and f_X is adjacent to each of the y_i 's and $w_X \sim y_1 \sim y_2 \sim y_3 \sim y_4$. Then $\mu(X) \leq \mu''$.*

Proof. By computation, we considered every such relevant boundary pair X (approximately 16×10^6 of them) and calculated $\mu(X)$. \square

6.4. Proving exponential decay. As in section 5, we will prove that $\Gamma_d(X)$ is exponentially small in d by defining a system of recursive equations. We will restrict attention to relevant boundary pairs; let Rel be the set of relevant boundary pairs. Say that a relevant boundary pair X is in \mathcal{N}_i^R (for $i \in \{0, \dots, 5\}$) if at most i of the neighbors of f_X are in R_X ; let $\Gamma_d^R = \max_{X \in \text{Rel}} \Gamma_d(X)$.

LEMMA 30. *Suppose $d > 1$. If $X \in \mathcal{N}_2^R$, then $\Gamma_d(X) \leq \Gamma_{d-1}^R/2$. If $X \in \mathcal{N}_3^R$, then $\Gamma_d(X) \leq 3\Gamma_{d-1}^R/4$. If $X \in \mathcal{N}_4^R$, then $\Gamma_d(X) \leq 20\Gamma_{d-1}^R/21$.*

Proof. If $X \in \mathcal{N}_2^R$, then by the definition of T_X , $\Gamma_d(X) \leq \nu(X) \cdot 2 \cdot \Gamma_{d-1}^R$. Then by Lemma 25 with $R' = \{f_X\}$ and Lemma 26, $\nu(X) \leq 1/4$. Similarly, if $X \in \mathcal{N}_3^R$, then $\Gamma_d(X) \leq \nu(X) \cdot 3 \cdot \Gamma_{d-1}^R \leq 3\Gamma_{d-1}^R/4$. Finally, suppose that exactly 4 neighbors of f_X are in R_X . Then $\Gamma_d(X) \leq \nu(X) \cdot 4 \cdot \Gamma_{d-1}^R$. Apply Lemma 25 where R' contains f_X and one of its neighbors in R_X . By Lemma 27, $\nu(X) \leq 5/21$. \square

Next, we define some subsets of Rel ; refer to Figure 5 to clarify these definitions. Let X be a relevant boundary pair.

- X is in U if there is a neighbor y_5 of f_X and of w_X that is not in R_X , and there is a neighbor $y_4 \neq w_X$ of f_X and of y_5 that is not in R_X .
- X is in V if there is a neighbor y_5 of f_X and of w_X that is not in R_X .
- X is in W if there is a neighbor y_5 of f_X and of w_X in R_X , and a neighbor $z_{11} \neq f_X$ of w_X and of y_5 that is not in R_X .

Let $U_d = \max_{X \in U} \Gamma_d(X)$, $V_d = \max_{X \in V} \Gamma_d(X)$, and $W_d = \max_{X \in W} \Gamma_d(X)$. The following lemma follows from the definition of T_X and the geometry of the lattice.

LEMMA 31. *Suppose that $q = 10$ and G is the triangular lattice. Suppose $d > 1$. Then*

$$\begin{aligned} \Gamma_d^R &\leq \max(\mu''(\Gamma_{d-1}^R + 2V_{d-1} + 2W_{d-1}), 20\Gamma_{d-1}^R/21), \\ U_d &\leq \max(\mu'(2V_{d-1} + W_{d-1}), \Gamma_{d-1}^R/2), \\ V_d &\leq \max(\mu''(2V_{d-1} + 2W_{d-1}), 3\Gamma_{d-1}^R/4), \text{ and} \\ W_d &\leq \max(\mu''(\Gamma_{d-1}^R + V_{d-1} + 2W_{d-1} + U_{d-1}), 20\Gamma_{d-1}^R/21). \end{aligned}$$

Proof. Suppose $X \in \text{Rel}$. If $X \in \mathcal{N}_4^R$, then $\Gamma_d(X) \leq 20\Gamma_{d-1}^R/21$ by Lemma 30. Otherwise an examination of Figure 5 reveals that

$$\Gamma_d(X) \leq \nu(X) (\Gamma_{d-1}^R + 2V_{d-1} + 2W_{d-1}).$$

The two instances of V_{d-1} correspond to y_1 and y_5 in the picture (since w_X is not in R_X), the two instances of W_{d-1} correspond to y_2 and y_4 , and the instance of Γ_{d-1}^R corresponds to y_3 . Apply Lemma 25 where R' is the set containing f_X and the vertices y_1, y_2, y_3 , and y_4 from Figure 5. By Lemma 29, $\nu(X) \leq \mu''$. This proves the upper bound on Γ_d^R .

Suppose $X \in U$. If $X \in \mathcal{N}_2^R$, then $\Gamma_d(X) \leq \Gamma_{d-1}^R/2$ by Lemma 30. Otherwise

$$\Gamma_d(X) \leq \nu(X) (2V_{d-1} + W_{d-1}).$$

As in the upper bound on Γ_d^R , one instance of V_{d-1} corresponds to y_1 and one instance of W_{d-1} corresponds to y_2 . An examination of Figure 5 reveals that, since $X \in U$, y_3 corresponds to V_{d-1} . Apply Lemma 25, where R' is the set containing f_X and the vertices y_1, y_2 , and y_3 from Figure 5; by Lemma 28, $\nu(X) \leq \mu'$. This proves the upper bound on U_d .

Suppose $X \in V$. If $X \in \mathcal{N}_3^R$, then $\Gamma_d(X) \leq 3\Gamma_{d-1}^R/4$ by Lemma 30. Otherwise

$$\Gamma_d(X) \leq \nu(X) (2V_{d-1} + 2W_{d-1}).$$

This is the same as the upper bound on Γ_d^R except that, since $X \in V$, y_3 corresponds to W_{d-1} and y_4 to V_{d-1} . Apply Lemma 25, where R' is the set containing f_X and the vertices y_1, y_2, y_3 , and y_4 from Figure 5. By Lemma 29, $\nu(X) \leq \mu''$. This proves the upper bound on V_d .

Suppose $X \in W$. If $X \in \mathcal{N}_4^R$, then $\Gamma_d(X) \leq 20\Gamma_{d-1}^R/21$ by Lemma 30. Otherwise

$$\Gamma_d(X) \leq \nu(X) (\Gamma_{d-1}^R + V_{d-1} + 2W_{d-1} + U_{d-1}).$$

This is the same as the upper bound on Γ_d^R except that, since $X \in W$, y_5 corresponds to U_{d-1} . Apply Lemma 25, where R' is the set containing f_X and the vertices y_1, y_2, y_3 , and y_4 from Figure 5; by Lemma 29, $\nu(X) \leq \mu''$. This proves the upper bound on W_d . \square

Here is the analogue of Lemma 18.

LEMMA 32. *Let $\zeta = 0.001$. Suppose that $q = 10$ and G is the triangular lattice. For every relevant boundary pair X and every $d \geq 1$, $\Gamma_d(X) \leq (1 - \zeta)^d$.*

Proof. Let $u = 15/26$, $v = 31/40$, and $w = 21/22$. We will prove by induction on d that $\Gamma_d^R \leq (1 - \zeta)^d$, $U_d \leq u(1 - \zeta)^d$, $V_d \leq v(1 - \zeta)^d$, and $W_d \leq w(1 - \zeta)^d$. For the base case, $d = 1$, note that for any relevant boundary pair X , $\Gamma_1(X) \leq \nu(X)$. Now apply Lemma 25 with $R' = \{f_X\}$ and by Lemma 26, we find that $\nu(X) \leq 1/4$. Thus, $\Gamma_1(x) \leq \min\{u, v, w, 1\}(1 - \zeta)$.

The inductive step follows directly from Lemma 31 since the following inequalities hold: $\mu''(1 + 2v + 2w) \leq 1 - \zeta$, $20/21 \leq 1 - \zeta$, $\mu'(2v + w) \leq u(1 - \zeta)$, $1/2 \leq u(1 - \zeta)$, $\mu''(2v + 2w) \leq v(1 - \zeta)$, $3/4 \leq v(1 - \zeta)$, $\mu''(1 + v + 2w + u) \leq w(1 - \zeta)$, and $20/21 \leq w(1 - \zeta)$. \square

Unlike Lemma 18, Lemma 32 applies just to relevant boundary pairs, so we have to finish up the proof of strong spatial mixing. Following the proof of Lemma 19, we obtain the following lemma.

LEMMA 33. *Let $\zeta = 0.001$. Suppose that $q = 10$ and G is the triangular lattice. Then for every relevant boundary pair X there exists a coupling Ψ of π_{B_X} and $\pi_{B'_X}$ such that, for all $f \in R_X$,*

$$\mathbb{E}[1_{\Psi, f}] \leq \frac{1}{\zeta}(1 - \zeta)^{d(f, s_X)}.$$

Furthermore,

$$\sum_{f \in R_X} \mathbb{E}[1_{\Psi, f}] \leq \frac{1 - \zeta}{\zeta}.$$

6.5. Vertex discrepancies and strong spatial mixing. The proof of strong spatial mixing (Theorem 7) is similar to the proof on section 4. The only extra problem is showing that the boundary pairs created in Lemma 20 are actually relevant boundary pairs (so that we can apply Lemma 33). This detail complicates the proof of the theorem, so we give a new version of the lemma.

LEMMA 34. *Let $\zeta = 0.001$. Suppose that $q = 10$ and G is the triangular lattice. For every vertex-boundary pair X there is a coupling Ψ of π_{B_X} and $\pi_{B'_X}$ such that, for all $f \in R$,*

$$\mathbb{E}[1_{\Psi, f}] \leq 10 \frac{1}{\zeta(1 - \zeta)}(1 - \zeta)^{d(f, v_X)}.$$

Furthermore,

$$\sum_{f \in R_X} \mathbb{E}[1_{\Psi, f}] \leq \frac{10(1 - \zeta)}{\zeta}.$$

Proof. First suppose that v_X has a neighbor $y \notin R_X$. (This case is straightforward and is like the proof of Lemma 20.) Let e_1, \dots, e_k be the boundary edges of R_X that are adjacent to v_X . Label these clockwise so that there is at least one nonboundary edge between e_k and e_1 . (The point here is that e_i and e_j are adjacent only if i and j differ by 1.) Let X_i be the relevant boundary pair consisting of the region R_X , the distinguished edge e_i , a coloring B of the boundary of R_X that agrees with \mathcal{B}_X except that edges e_1, \dots, e_{i-1} are colored with color $\mathcal{B}'_X(v_X)$ and e_i, \dots, e_k are colored with color $\mathcal{B}_X(v_X)$, and a coloring B' that is the same as B except that it colors e_i with color $\mathcal{B}'_X(v_X)$. We construct a coupling of $\pi_{\mathcal{B}_X}$ and $\pi_{\mathcal{B}'_X}$ by composing couplings Ψ_1, \dots, Ψ_k of X_1, \dots, X_k . Now

$$\mathbb{E}[1_{\Psi, f}] \leq \sum_{i=1}^k \mathbb{E}[1_{\Psi_i, f}] \leq 5 \frac{1}{\zeta} (1 - \zeta)^{d(f, v_X)}.$$

Now we must deal with the case in which all neighbors of v_X are in R_X . A technical detail arises here because the natural induced boundary pairs are not all relevant. Let y be any neighbor of v_X . Let Ψ' be any coupling of $\pi_{\mathcal{B}_X}$ and $\pi_{\mathcal{B}'_X}$. Let (C, C') be the random variable corresponding to the pair of colorings in $S(\mathcal{B}_X) \times S(\mathcal{B}'_X)$ drawn from Ψ . We will choose the color of y in C and C' according to Ψ' . To complete the construction of Ψ , for every pair (c, c') , we will let $\mathcal{B}_X(c)$ denote the vertex-boundary of $R_X - \{y\}$ which agrees with \mathcal{B}_X except that y is colored c , and we will let $\mathcal{B}'_X(c')$ denote the vertex-boundary of $R_X - \{y\}$ which agrees with \mathcal{B}'_X except that y is colored c' . We will construct a coupling of $\mathcal{B}_X(c)$ and $\mathcal{B}'_X(c')$ by composing the couplings of up-to-10 relevant boundary pairs (these boundary pairs correspond to discrepancies on the 5 boundary edges of vertex v_X and the up-to-5 boundary edges on vertex y). The $(1 - \zeta)$ in the denominator comes from the fact that the distance from a vertex f to the discrepancy edge may be one less than $d(f, v_X)$. \square

Lemma 34 provides an ε -coupling cover for $\varepsilon = \frac{\zeta}{1-\zeta} \frac{6}{10}$. Since the lattice is neighborhood-amenable, we obtain Corollary 11.

6.6. Extensions. Using techniques similar to those presented in section 6, we can give an alternative proof to the result of [1]—strong spatial mixing for 6-colorings of the rectangular lattice. The amount of computation in the alternative proof and the proof in section 6 can be reduced by applying some of the techniques from Lemma 16. Our technique can also be applied to other lattices, for example, some of the others studied by Salas and Sokal [27].

7. Rapid mixing. Now we prove Theorem 8, showing that for neighborhood-amenable graphs, our strong spatial mixing proof implies rapid mixing. It is known that strong spatial mixing implies rapid mixing in such cases (see [13, 22, 31]) but existing proofs seem to be written for \mathbb{Z}^d , so we add this section for completeness.

THEOREM 8. *Let G denote an infinite neighborhood-amenable graph with maximum degree Δ . Let R be a finite subgraph of G with $|R| = n$ and $\mathcal{B}(R)$ denote a coloring of $\partial(R)$ using the colors $Q \cup \{0\}$. (We assume that $q \geq \Delta + 2$.)*

Suppose there exists $\varepsilon > 0$ such that G has an ε -coupling cover. Then the Glauber dynamics Markov chain on $S(\mathcal{B}(R))$ is rapidly mixing and $\tau(\delta) \in O(n(n + \log \frac{1}{\delta}))$.

We use the method of path coupling to prove this theorem, approaching our result indirectly through the use of Markov chain comparison. We give a brief review of the path-coupling method in the next section, then proceed with the first step in our analysis for graphs that satisfy the hypotheses of Theorem 8. In section 7.2 we first examine an auxiliary Markov chain which allows recoloring of a slightly larger set of

vertices in a single recoloring step. We show this new chain mixes in time $O(n \log n)$. Markov chain comparison is reviewed in section 7.3, and then the second part of the proof of Theorem 8 is presented in section 7.4.

7.1. Path coupling. Coupling is a popular method for analyzing mixing times of Markov chains. A (Markovian) coupling for a Markov chain \mathcal{M} with state space Ω is a stochastic process (X_t, Y_t) on $\Omega \times \Omega$ such that each of (X_t) and (Y_t) , considered marginally, is a faithful copy of \mathcal{M} . The coupling lemma (see, for example, Aldous [2]) states that the total variation distance of \mathcal{M} at time t is bounded above by $\Pr(X_t \neq Y_t)$. The *path-coupling* method, introduced in [4], is a powerful method for finding couplings. The idea is that one can find a coupling on a subset U of $\Omega \times \Omega$ and extend this to a coupling on $\Omega \times \Omega$. The following theorem, adapted from [12], summarizes the path-coupling method.

THEOREM 35 (see [4, 12]). *Let U be a relation $U \subseteq \Omega^2$ such that U has transitive closure Ω^2 . Let $\phi : U \rightarrow \{0, 1, 2, \dots\}$ be a “proximity function” defined on pairs in U . We use ϕ to define a function Φ on Ω^2 as follows: For each pair $(\omega, \omega') \in \Omega^2$, let*

$$\Phi(\omega, \omega') = \min_{\omega_0, \dots, \omega_k} \sum_{i=0}^{k-1} \phi(\omega_i, \omega_{i+1}),$$

where the minimum is over all paths $\omega = \omega_0, \dots, \omega_k = \omega'$ such that, for all $i \in [0, k-1]$, $(\omega_i, \omega_{i+1}) \in U$. Let (X_t, Y_t) be a coupling for \mathcal{M} defined over all pairs in U . Suppose for this coupling there is $\beta < 1$ such that for all $(\sigma_1, \sigma_2) \in U$ we have

$$\mathbb{E}[\Phi(X_{t+1}, Y_{t+1}) \mid (X_t, Y_t) = (\sigma_1, \sigma_2)] \leq \beta \Phi(\sigma_1, \sigma_2).$$

Let D be the maximum value that Φ achieves on Ω^2 . Then

$$\tau(\delta) \leq \frac{\ln(D/\delta)}{1 - \beta}.$$

7.2. Proof of rapid mixing (Part I). Our goal is to sample from $S(\mathcal{B})$ the set of proper colorings of R consistent with the boundary coloring, uniformly at random, using the single-vertex Glauber dynamics Markov chain. To do this, we first define another Markov chain that corresponds to heat-bath dynamics on small subregions of R . As we defined in section 1.1, for a vertex $f \in G$ and a nonnegative integer d we let $Ball_d(f)$ denote the set of vertices that are at most distance d from f .

Now consider a problem instance of R and \mathcal{B} . For a fixed $d \geq 0$ (to be specified later) and a vertex $f \in G$, let $R_f = Ball_d(f) \cap R$. Further, let

$$R^* = \{f \in G \mid R_f \neq \emptyset\}.$$

Then \mathcal{M}_d is the heat-bath Markov chain with state space $S(\mathcal{B})$ and the following transitions: \mathcal{M}_d makes a transition from a state $\sigma \in S(\mathcal{B})$ by choosing a vertex $f \in R^*$ uniformly at random. Let \mathcal{B}_f be the coloring of ∂R_f induced by σ and \mathcal{B} . To make the transition from σ , recolor the vertices in R_f by sampling from $\pi_{\mathcal{B}_f}$, the uniform distribution of colorings on the region R_f induced by \mathcal{B}_f .

Since $Ball_0(f) = \{f\}$, Glauber dynamics is \mathcal{M}_0 . However, in order to prove rapid mixing of \mathcal{M}_0 we first demonstrate rapid mixing of the chain \mathcal{M}_d for some constant d . Then we use the comparison method (see section 7.3 below) to infer rapid mixing of \mathcal{M}_0 .

Proof of Theorem 8 (Part I). Path coupling is used to prove rapid mixing of the Markov chain \mathcal{M}_d . First, we specify the value of d that we use.

Fix an $\varepsilon > 0$ for which G has an ε -coupling cover as guaranteed in the hypothesis of Theorem 8. Recalling Definition 2, since G is neighborhood-amenable, we can find d such that

$$T_d = \sup_v \frac{|\partial Ball_d(v)|}{|Ball_d(v)|} \leq \frac{\varepsilon}{\varepsilon + \Delta}.$$

In this setting, the distance measure we use is Hamming distance. Because of this, we use the standard approach of taking the set Ω in Theorem 35 to be the set of *all* (proper and improper) q -colorings of the region R . We take U to be the set of pairs of colors that differ at a single vertex. Consider two colorings σ and θ in U with Hamming distance 1, i.e., σ and θ are two (not necessarily proper) colorings of R that disagree at a single vertex v . We describe a coupled transition from the pair (σ, θ) to a new pair of colorings (σ', θ') . In this coupling, we choose the same vertex f for the transition $\sigma \rightarrow \sigma'$ that we choose for the transition $\theta \rightarrow \theta'$. Note that while σ and θ may not be proper colorings of R , a transition $\sigma \rightarrow \sigma'$ is allowed only if σ' is “not more improper” than σ , and similarly for a transition $\theta \rightarrow \theta'$. In other words, having chosen a vertex f , we recolor the “window” R_f using a proper coloring of that window (conditioned, of course, on its induced boundary coloring).

First, note that by construction of R^* , we have $Ball_d(v) \subseteq R^*$. For each vertex $f \in Ball_d(v)$, if f is chosen in the chain \mathcal{M}_d , we can ensure that $\sigma' = \theta'$ by choosing the same coloring to recolor R_f . Thus there are $|Ball_d(v)|$ ways to decrease the Hamming distance by one.

If the chosen vertex f is far from v , in the sense that $v \notin \partial R_f$, then we will couple the transitions by again choosing the same recoloring for the region R_f . This ensures that σ' and θ' disagree only at v so they still have Hamming distance 1.

We now calculate an upper bound on how much the distance can increase in one step of the coupling. This can happen only if we choose some vertex f such that $v \in \partial R_f$. With this in mind we define $\mathcal{H}_d(v) = \{f \in R^* : v \in \partial R_f\}$; $|\mathcal{H}_d(v)|$ is an upper bound on the number of vertices whose selection can increase the distance in the new pair (σ', θ') . Let \mathcal{B}_1 be the coloring of ∂R_f induced from the coloring σ and let \mathcal{B}_2 be that induced from θ . Then \mathcal{B}_1 and \mathcal{B}_2 differ solely at the vertex v . The ε -coupling cover in the hypothesis of the theorem guarantees we can construct a coupling that allows us to choose a pair (σ', θ') of proper colorings so that the expected Hamming distance increases by at most Δ/ε .

Adding it all up, we see the expected Hamming distance between σ' and θ' after one step of the coupling is at most

$$1 - \frac{|Ball_d(v)|}{|R^*|} + \frac{|\mathcal{H}_d(v)|}{|R^*|} \frac{\Delta}{\varepsilon} = 1 - \frac{|Ball_d(v)|\varepsilon - |\mathcal{H}_d(v)|\Delta}{|R^*|\varepsilon}.$$

From the choice of d (using the neighborhood-amenable property of G), and using $\mathcal{H}_d(v) \subseteq \partial Ball_d(v)$, we have

$$|Ball_d(v)|\varepsilon - |\mathcal{H}_d(v)|\Delta \geq |\partial Ball_d(v)|(\varepsilon + \Delta) - |\partial Ball_d(v)|\Delta = |\partial Ball_d(v)|\varepsilon \geq \varepsilon.$$

Thus we have $\mathbb{E}[\Phi(\sigma', \theta')] \leq 1 - 1/|R^*|$.

So for Theorem 35, we can take $\beta = 1 - 1/|R^*|$. Since we use Hamming distance, we have that $D = n$. Using the facts that $d \in O(1)$ and that $\Delta \in O(1)$ (since Δ

is a universal bound on maximum degree for any finite subgraph of G), we see that $1/(1 - \beta) = |R^*| \in O(n)$. Using Theorem 35 we conclude that $\tau_{\mathcal{M}_d}(\delta) \in O(n \log(\frac{n}{\delta}))$.

The final step to get the desired result about \mathcal{M}_0 uses the method of comparing Markov chains. We review this method below, then continue with the proof of Theorem 8 after that.

7.3. The comparison method. In the previous section we showed rapid mixing of \mathcal{M}_d on the set of all (proper and improper) colorings. This implies that \mathcal{M}_d mixes rapidly on the set of proper colorings. In this section we will compare the mixing times of \mathcal{M}_d and \mathcal{M}_0 on the set of proper colorings. We use the method of Diaconis and Saloff-Coste [7]. We provide definitions in the context of these two coloring Markov chains. P_d (resp., P_0) will be used to denote the transition matrix for the chain \mathcal{M}_d (resp., \mathcal{M}_0).

For $i \in \{0, d\}$, let E_i be the set of pairs of distinct colorings (σ, θ) with $P_i(\sigma, \theta) > 0$. We will sometimes refer to the members of E_i as “edges” because they are edges in the transition graph of \mathcal{M}_i . For every edge $(\sigma, \theta) \in E_d$, let $\mathcal{P}_{\sigma, \theta}$ be the set of paths from σ to θ using transitions of \mathcal{M}_0 . More formally, let $\mathcal{P}_{\sigma, \theta}$ be the set of paths $\gamma = (\sigma = \sigma_0, \sigma_1, \dots, \sigma_k = \theta)$ such that

1. each (σ_i, σ_{i+1}) is in E_0 , and
2. each $(\sigma', \theta') \in E_0$ appears at most once on γ .

We write $|\gamma|$ to denote the length of path γ . So, for example, if $\gamma = (\sigma_0, \dots, \sigma_k)$ we have $|\gamma| = k$. Let $\mathcal{P} = \cup_{(\sigma, \tau) \in E_d} \mathcal{P}_{\sigma, \tau}$.

A *flow* is a function ϕ from \mathcal{P} to the interval $[0, 1]$ such that for every $(\sigma, \theta) \in E_d$,

$$(7) \quad \sum_{\gamma \in \mathcal{P}_{\sigma, \theta}} \phi(\gamma) = P_d(\sigma, \theta) \pi_{\mathcal{B}}(\sigma).$$

For every $(\sigma', \theta') \in E_0$, the *congestion* of edge (σ', θ') in the flow ϕ is the quantity

$$A_{\sigma', \theta'}(\phi) = \frac{1}{\pi_{\mathcal{B}}(\sigma') P_0(\sigma', \theta')} \sum_{\gamma \in \mathcal{P}: (\sigma', \theta') \in \gamma} |\gamma| \phi(\gamma).$$

The *congestion* of the flow is the quantity

$$A(\phi) = \max_{(\sigma', \theta') \in E_0^*} A_{\sigma', \theta'}(\phi).$$

A proof of the following theorem can be found in [11, Observation 13]. This theorem is similar to Proposition 4 of Randall and Tetali [26] except that the latter requires the eigenvalues of the transition matrices to be nonnegative. Both results are based closely on the ideas of Aldous [2], Diaconis and Stroock [8], and Sinclair [28].

THEOREM 36. *Suppose that ϕ is a flow. Let $c = \min_{\sigma} P_0(\sigma, \sigma)$ and note that $c \geq 1/q$. Then for any $0 < \delta' < \frac{1}{2}$*

$$\tau(\mathcal{M}_0, \delta) \leq \max \left\{ A(\phi) \left[\frac{\tau(\mathcal{M}_d, \delta')}{\ln \frac{1}{2\delta'}} + 1 \right], \frac{1}{2c} \right\} \ln \frac{1}{\delta \cdot \pi_{\min}},$$

where $\pi_{\min} = \min_{\sigma} \pi_{\mathcal{B}}(\sigma)$.

We continue with the proof of Theorem 8 in the next section.

7.4. Proof of rapid mixing (Part II). Suppose we take $\delta' = 1/n$ and use the upper bound from the first part of the proof of Theorem 8. We then have $\tau(\mathcal{M}_d, \delta') \in O(n \log n)$. We now construct a flow ϕ such that $A(\phi) \in O(1)$, and then Theorem 36 gives

$$\tau(\mathcal{M}_0, \delta) \leq O(1) \cdot O(n) \cdot \ln \frac{1}{\delta \cdot \pi_{\min}}.$$

This yields Theorem 8 since $\ln(1/\pi_{\min}) \in O(n)$.

Proof of Theorem 8 (Part II).

Constructing a flow.

Consider a problem instance consisting of a nonempty region R with $|R| = n$ and a coloring \mathcal{B} of ∂R . We will now construct a flow ϕ .

For every pair $(\sigma, \theta) \in E_d$, we fix some vertex f such that $Ball_d(f)$ contains all the vertices on which σ and θ differ. Then we fix a canonical ordering on these vertices where they differ, say v_1, \dots, v_m .

Let $\gamma_{\sigma, \theta} \in \mathcal{P}_{\sigma, \theta}$ be the canonical path from σ to θ , constructed as follows:

- Update the vertices v_1, \dots, v_m in order.
- In order to update a given vertex v_i :
 - If any neighbors of v_i have color $\theta(v_i)$, recolor these with the lexicographically first available color. (Note that these neighbors do not have their final color in θ .)
 - Recolor v_i with color $\theta(v_i)$.

Assign all of the flow from σ to θ to path $\gamma_{\sigma, \theta}$. That is, set $\phi(\gamma_{\sigma, \theta}) = P_d(\sigma, \theta) \pi_{\mathcal{B}}(\sigma)$.

Bounding $A(\phi)$.

We show that $A(\phi) \in O(1)$, which completes the proof of Theorem 8.

Let σ' and θ' , where $(\sigma', \theta') \in E_0$, be colorings that disagree on vertex x . Now

$$A_{\sigma', \theta'}(\phi) = \frac{1}{\pi_{\mathcal{B}}(\sigma') P_0(\sigma', \theta')} \left(\sum_{\substack{(\sigma, \theta) \in E_d \\ (\sigma', \theta') \in \gamma_{\sigma, \theta}}} |\gamma_{\sigma, \theta}| P_d(\sigma, \theta) \pi_{\mathcal{B}}(\sigma) \right).$$

Since $\pi_{\mathcal{B}}$ is uniform and all of the path lengths are $O(1)$, this simplifies to

$$A_{\sigma', \theta'}(\phi) \leq O(1) \times \left(\sum_{\substack{(\sigma, \theta) \in E_d \\ (\sigma', \theta') \in \gamma_{\sigma, \theta}}} \frac{P_d(\sigma, \theta)}{P_0(\sigma', \theta')} \right).$$

To see that this sum is $O(1)$ note that there are only $O(1)$ pairs (σ, θ) in the summation (this holds since σ and θ agree with σ' except in a constant-sized ball around x), since $\sigma \neq \theta$, $P_d(\sigma, \theta) \in O(1/n)$. Finally, $P_0(\sigma', \theta') \in \Omega(1/n)$.

7.5. Neighborhood-amenability—How restrictive is it? Theorem 8 applies to graphs that are neighborhood-amenable. This condition, while sufficient, is not necessary. The theorem could be extended to a larger class of graphs. The relevant issue is to balance the number of “good” transitions that decrease the distance between the pair with Hamming distance 1 with the number of “bad” transitions that increase the distance (of course, how much the distance increases from any bad transition also matters). There are other similar conditions that we might require from our graph to prove rapid mixing.

Instead of studying these conditions here, we show that neighborhood-amenability is applicable fairly widely. We do this by defining an alternative natural condition and showing that it implies neighborhood-amenability.

DEFINITION 37. For a vertex v of G , let $N_d(v)$ denote the set of vertices that are at distance d from v , and let $n_d(v) = |N_d(v)|$. (Note that $n_0(v) = 1$.)

We say that G is uniformly subexponential if there exists a function $\kappa(d)$ such that

1. for all $b > 1$, $\kappa(d) \in o(b^d)$, and
2. there exist $c_2 \geq c_1 > 0$ such that for all $v \in G$, $c_1\kappa(d) \leq n_d(v) \leq c_2\kappa(d)$.

As stated, the condition of being uniformly subexponential implies neighborhood-amenability as we show below. We first state a lemma that we use to prove this claim.

LEMMA 38. Let $\{a_i\}_{i \geq 0}$ be a sequence of positive numbers. Suppose that $a_d > \alpha(a_{d-1} + a_{d-2} + \dots + a_0)$ for all $d \geq 1$. Then $a_d > a_0\alpha(1 + \alpha)^{d-1}$ for $d \geq 1$.

Proof. We prove this by induction, where the base case (with $d = 1$) is obvious from the condition imposed on the sequence. Then

$$\begin{aligned} a_d &> \alpha(a_{d-1} + a_{d-2} + \dots + a_1 + a_0) \\ &> \alpha(a_0\alpha(1 + \alpha)^{d-2} + a_0\alpha(1 + \alpha)^{d-3} + \dots + a_0\alpha + a_0) \quad (\text{by inductive assumption}) \\ &= a_0\alpha \left(\alpha \left((1 + \alpha)^{d-2} + (1 + \alpha)^{d-3} + \dots + 1 \right) + 1 \right) \\ &= a_0\alpha \left(\alpha \frac{(1 + \alpha)^{d-1} - 1}{\alpha} + 1 \right) \\ &= a_0\alpha(1 + \alpha)^{d-1}. \quad \square \end{aligned}$$

LEMMA 39. Suppose that G is uniformly subexponential. Then G is neighborhood-amenable.

Proof. We apply the contrapositive of Lemma 38 to the sequence of numbers $\kappa(0), \kappa(1), \dots$. This means that given $\varepsilon > 0$, there is a d such that $\kappa(d + 1) \leq \varepsilon(\kappa(d) + \dots + \kappa(0))$. (Otherwise, if $\kappa(d + 1) > \varepsilon(\kappa(d) + \dots + \kappa(0))$ for all d , then the lemma says that $\kappa(d + 1) > \kappa(0)\varepsilon(1 + \varepsilon)^d$ for all $d \geq 0$. Clearly this would violate condition 1 in the definition of “uniformly subexponential.”)

Thus, for any vertex $v \in G$ we have, using condition 2 in Definition 37,

$$\begin{aligned} |\partial Ball_d(v)| = n_{d+1}(v) &\leq c_2\kappa(d + 1) \\ &\leq c_2 \cdot \varepsilon(\kappa(d) + \dots + \kappa(0)) \\ &\leq \frac{c_2}{c_1} \cdot \varepsilon(n_d(v) + \dots + n_0(v)) = \frac{c_2}{c_1} \cdot \varepsilon|Ball_d(v)|. \end{aligned}$$

Hence $T_d = \sup_v \frac{|\partial Ball_d(v)|}{|Ball_d(v)|} \leq \frac{c_2}{c_1} \cdot \varepsilon$. Since $\varepsilon > 0$ is arbitrary, this implies that G is neighborhood-amenable. \square

One could think of other conditions that would imply neighborhood-amenability or even different conditions for which a similar proof of rapid mixing such as the one we gave in Theorem 8 could be demonstrated. If we are dealing with a graph that is vertex-transitive, for example, checking whether it is uniformly-subexponential or not provides a relatively straightforward method to determine if it is neighborhood-amenable.

Readers should consult [30, 31] for further discussion about conditions under which one could demonstrate rapid mixing of Markov chains for sampling proper colorings.

Acknowledgments. We thank Martin Dyer and Eric Vigoda for useful discussions. We are also grateful to a very helpful anonymous referee.

REFERENCES

- [1] D. ACHLIOPTAS, M. MOLLOY, C. MOORE, AND F. VAN BUSSEL, *Sampling grid colorings with fewer colors*, in Proceedings of the 6th Annual Latin American Theoretical Informatics (LATIN 2004), Buenos Aires, Argentina, pp. 80–89.
- [2] D. ALDOUS, *Random walks on finite groups and rapidly mixing Markov chains*, Séminaire de Probabilités XVII 1081/1082, A. Dold and B. Eckmann, eds., Lecture Notes in Math., 986, Springer-Verlag, Berlin, 1983, pp. 243–297.
- [3] G. BRIGHTWELL AND P. WINKLER, *Graph homomorphisms and phase transitions*, J. Combin. Theory Ser. B, 77 (1999), pp. 221–262.
- [4] R. BUBLEY AND M. DYER, *Path coupling: A technique for proving rapid mixing in Markov chains*, in Proceedings of the 38th Annual IEEE Symposium on Foundations of Computer Science (FOCS 1997), Miami Beach, FL, pp. 223–231.
- [5] R. BUBLEY, M. DYER, C. GREENHILL, AND M. JERRUM, *On approximately counting colorings of small degree graphs*, SIAM J. Comput., 29 (1999), pp. 387–400.
- [6] F. CESI, *Quasi-factorization of the entropy and logarithmic Sobolev inequalities for Gibbs random fields*, Probab. Theory Related Fields, 120 (2001), pp. 569–584.
- [7] P. DIACONIS AND L. SALOFF-COSTE, *Comparison theorems for reversible Markov chains*, Ann. Appl. Probab., 3 (1993), pp. 696–730.
- [8] P. DIACONIS AND D. STROOCK, *Geometric bounds for eigenvalues of Markov chains*, Ann. Appl. Probab., 1 (1991), pp. 36–61.
- [9] M. DYER AND A. FRIEZE, *Randomly coloring graphs with lower bounds on girth and maximum degree*, Random Structures Algorithms, 23 (2003), pp. 167–179.
- [10] M. DYER, A. FRIEZE, T. P. HAYES, AND E. VIGODA, *Randomly coloring constant degree graphs*, in Proceedings of the 45th Annual IEEE Symposium on Foundations of Computer Science (FOCS 2004), Rome, pp. 582–589.
- [11] M. DYER, L.A. GOLDBERG, M. JERRUM, AND R. MARTIN, *Markov chain comparison*, available online at <http://arxiv.org/abs/math.PR/0410331>.
- [12] M. DYER AND C. GREENHILL, *Random walks on combinatorial objects*, in Surveys in Combinatorics, 1999, London Math. Soc. Lecture Note Ser. 267, J. D. Lamb and D. A. Preece, eds., Cambridge University Press, Cambridge, UK, 1999, pp. 101–136.
- [13] M. DYER, A. SINCLAIR, E. VIGODA, AND D. WEITZ, *Mixing in time and space for lattice spin systems: A combinatorial view*, Random Structures Algorithms, 24 (2004), pp. 461–479.
- [14] A. FRIEZE AND J. VERA, *On randomly coloring locally sparse graphs*, preprint 2004.
- [15] L. A. GOLDBERG, R. MARTIN, AND M. PATERSON, *Random sampling of 3-colorings in \mathbb{Z}^2* , Random Structures Algorithms, 24 (2004), pp. 279–302.
- [16] T. HAYES, *Randomly coloring graphs of girth at least five*, in Proceedings of the 35th Annual ACM Symposium on Theory of Computing (STOC 2003), San Diego, CA, pp. 269–278.
- [17] T. HAYES AND E. VIGODA, *A nonMarkovian coupling for randomly sampling colorings*, in Proceedings of the 44th Annual IEEE Symposium on Foundations of Computer Science (FOCS 2003), Cambridge, MA, pp. 618–627.
- [18] T. HAYES AND E. VIGODA, *Coupling with the stationary distribution and improved sampling for colorings and independent sets*, in Proceedings of the 16th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2005), Vancouver, Canada, pp. 971–979.
- [19] M. JERRUM, *A very simple algorithm for estimating the number of k -colorings of a low-degree graph*, Random Structures Algorithms, 7 (1995), pp. 157–165.
- [20] C. KENYON, E. MOSSEL, AND Y. PERES, *Glauber dynamics on trees and hyperbolic graphs*, in Proceedings of the 42nd Annual IEEE Symposium on Foundations of Computer Science (FOCS 2001), Las Vegas, NV, pp. 568–578.
- [21] M. LUBY, D. RANDALL, AND A. J. SINCLAIR, *Markov chain algorithms for planar lattice structures*, SIAM J. Comput., 31 (2001), pp. 167–192.
- [22] F. MARTINELLI, *Lectures on Glauber dynamics for discrete spin models*, Lectures on Probability Theory and Statistics, Saint-Flour, 1997. Lecture Notes in Math. 1717, Springer, Berlin (1999), pp. 93–191.
- [23] F. MARTINELLI, A. SINCLAIR, AND D. WEITZ, *The Ising model on trees: Boundary conditions and mixing time*, in Proceedings of the 44th Annual IEEE Symposium on Foundations of Computer Science (FOCS 2003), Cambridge, MA, pp. 628–639.
- [24] F. MARTINELLI, A. SINCLAIR, AND D. WEITZ, *Fast mixing for independent sets, colorings and*

- other models on trees*, in Proceedings of the 15th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2004), New Orleans, pp. 456–465.
- [25] M. MOLLOY, *The Glauber dynamics on colorings of a graph with high girth and maximum degree*, SIAM J. Comput., 33 (2004), pp. 721–737.
 - [26] D. RANDALL AND P. TETALI, *Analyzing Glauber dynamics by comparison of Markov chains*, J. Math. Phys., 41 (2000), pp. 1598–1615.
 - [27] J. SALAS AND A. D. SOKAL, *Absence of phase transition for antiferromagnetic Potts models via the Dobrushin uniqueness theorem*, J. Statist. Phys., 86 (1997), pp. 551–579.
 - [28] A. SINCLAIR, *Improved bounds for mixing rates of Markov chains and multicommodity flow*, Combin. Probab. Comput., 1 (1992), pp. 351–370.
 - [29] E. VIGODA, *Improved bounds for sampling colorings*, J. Math. Phys., 41 (2000), pp. 1555–1569.
 - [30] D. WEITZ, *Combinatorial criteria for uniqueness of Gibbs measures*, Random Structures Algorithms, to appear.
 - [31] D. WEITZ, *Mixing in Time and Space for Discrete Spin Systems*, Ph.D. thesis, University of California, Berkeley, CA, 2004.

GENERAL MULTIPROCESSOR TASK SCHEDULING: APPROXIMATE SOLUTIONS IN LINEAR TIME*

KLAUS JANSEN[†] AND LORANT PORKOLAB[‡]

Abstract. We study the problem of scheduling n independent tasks on a set of m parallel processors, where the execution time of a task is a function of the subset of processors assigned to the task. For any fixed m , we propose a fully polynomial approximation scheme that for any fixed $\epsilon > 0$ finds a preemptive schedule of length at most $(1 + \epsilon)$ times the optimum in $O(n)$ time. We also discuss the nonpreemptive variant of the problem, and present for any fixed m a polynomial approximation scheme that computes an approximate solution of any fixed accuracy in linear time. In terms of the running time, this linear complexity bound gives a substantial improvement of the best previously known polynomial bound [J. Chen and A. Miranda, *SIAM J. Comput.*, 31 (2001), pp. 1–17].

Key words. approximation algorithms, scheduling problem, linear programming

AMS subject classifications. 68Q20, 68Q25, 90B35, 90C27, 90C39

DOI. 10.1137/S0097539799361737

1. Introduction. In classical scheduling theory, each task is processed by only one processor at a time. However, recently, due to the rapid development of parallel computer systems, new theoretical approaches have emerged to model scheduling on parallel architectures. One of these is scheduling multiprocessor tasks (see, e.g., [5, 8, 9]).

In this paper we address some multiprocessor scheduling problems, where a set of n tasks has to be executed by m processors such that each processor can work on at most one task at a time and a task can be (or may need to be) processed simultaneously by several processors. In the *dedicated* variant of this model, each task requires the simultaneous use of a prespecified set of processors. In the *parallel* variant, which is also called *nonmalleable* parallel task scheduling, the multiprocessor architecture is disregarded, and there is simply a size associated with each task which indicates that the task can be processed on any subset of processors of this size. In the *malleable* variant, each task can be executed on an arbitrary (nonempty) subset of processors, and the execution time of the task depends on the number of processors assigned to it. A generalization of the dedicated variant allows tasks to have a number of *alternative modes*, where each processing mode specifies a subset of processors and the task's execution time on that particular processor set. This problem is called *general multiprocessor task scheduling*.

Depending on the model, tasks can be preempted or not. In the *nonpreemptive* model, a task once started has to be completed without interruption. In the *pre-*

*Received by the editors September 23, 1999; accepted for publication (in revised form) July 7, 2005; published electronically December 8, 2005. This work was done while the authors were associated with the research institutes IDSIA Lugano and MPII Saarbrücken and were supported in part by the Swiss Office Fédéral de l'éducation et de la Science project 97.0315 titled "Platform" and by EU ESPRIT LTR project 20244 (ALCOM-IT). The extended abstract of a preliminary version of this paper appeared in *Proceedings of the 6th International Workshop on Algorithms and Data Structures (WADS '99)*.

<http://www.siam.org/journals/sicomp/35-3/36173.html>

[†]Institut für Informatik und praktische Mathematik, Universität zu Kiel, D-24098, Kiel, Germany (kj@informatik.uni-kiel.de).

[‡]Department of Computing, Imperial College, London SW7 2AZ, UK (porkolab@doc.ic.ac.uk).

emptive model, each task can be interrupted any time at no cost and restarted later, possibly on a different set of processors. The objective of the scheduling problems discussed in this paper is to minimize the makespan, i.e., the maximum completion time C_{max} . The nonpreemptive versions of malleable and nonmalleable parallel task scheduling have recently been discussed in several papers (see, e.g., [9, 15, 16]). The authors have also studied this problem [12] and proposed an approximation scheme that finds for any fixed $\epsilon > 0$ a nonpreemptive schedule of makespan at most $(1 + \epsilon)$ times the optimum (such a schedule is called an ϵ -approximate solution). The algorithm runs in linear time when the number of parallel processors is fixed. The dedicated and general variants of nonpreemptive (preemptive) scheduling for independent multiprocessor tasks on a fixed number of processors are denoted by $Pm|fix_j|C_{max}$ ($Pm|fix_j, pmtn|C_{max}$) and $Pm|set_j|C_{max}$ ($Pm|set_j, pmtn|C_{max}$), respectively.

Regarding the complexity, problems $P3|fix_j|C_{max}$ and $P3|set_j|C_{max}$ are strongly NP-hard [3, 4, 11], thus they do not have fully polynomial-time approximation schemes, unless $P=NP$. Recently, Amoura et al. [1] developed a polynomial-time approximation scheme for $Pm|fix_j|C_{max}$. For the general problem $Pm|set_j|C_{max}$, Bianco et al. [2] presented an approximation algorithm whose approximation ratio is bounded by m . Later Chen and Lee [6] improved their algorithm by achieving an approximation ratio $\frac{m}{2} + \epsilon$. Until very recently, this was the best approximation result for the problem, and it was not known whether there is a polynomial-time approximation scheme or even a polynomial-time approximation algorithm with a constant approximation guarantee. Independently from our work presented here, Chen and Miranda [7] have recently proposed a polynomial-time approximation scheme for the problem. The running time of their approximation scheme is $O(n^{\lambda_{m,\epsilon} + j_{m,\epsilon} + 1})$, where $\lambda_{m,\epsilon} = (2j_{m,\epsilon} + 1)B_m m$ and $j_{m,\epsilon} \leq (3mB_m + 1)^{\lceil m/\epsilon \rceil}$ with $B_m \leq m!$ denoting the m th Bell number. In this paper, we propose another polynomial-time approximation scheme for $Pm|set_j|C_{max}$ that computes an ϵ -approximate solution in $O(n)$ time for any fixed positive accuracy ϵ . In terms of the running time this gives a substantial improvement of the previously mentioned result [7] and also answers an open question of that paper by providing an approach that is not based on dynamic programming.

It is known that the preemptive variant $Pm|set_j, pmtn|C_{max}$ of the problem can be solved in polynomial time [2] by formulating it as a linear program with n constraints and n^m variables and computing an optimal solution by using any polynomial-time linear programming algorithm. Even though (for any fixed m), the running time in this approach is polynomial in n , the degree of this polynomial depends linearly on m . Therefore it is natural to ask whether there are more efficient algorithms for $Pm|set_j, pmtn|C_{max}$ (of running time, say, for instance, $O(n)$ or $O(n^c)$ with an absolute constant c) that compute exact or approximate solutions. In this paper we focus on approximate solutions and present a fully polynomial approximation scheme for $Pm|set_j, pmtn|C_{max}$ that finds an ϵ -approximate solution in $O(n)$ time for any fixed positive accuracy ϵ . This result also shows that as long as approximate solutions (of any fixed positive accuracy) are concerned, linear running time can be achieved for the problem, but it leaves open the question (which also partly motivated this work) of whether there exists a linear-time algorithm for computing an optimal solution of $Pm|set_j, pmtn|C_{max}$. This question was answered in an affirmative way for the dedicated variant $Pm|fix_j, pmtn|C_{max}$ of the problem by Amoura et al. [1].

2. Preemptive scheduling. In this section we consider the preemptive version $Pm|pmtn, set_j|C_{max}$ of the general multiprocessor task scheduling problem. Suppose there are a set of tasks $\mathcal{T} = \{T_0, \dots, T_{n-1}\}$ and a set of processors $M = \{1, \dots, m\}$.

Each task T_j has an associated function $t_j : 2^M \rightarrow Q^+ \cup \{+\infty\}$ that gives the execution time $t_j(S)$ of task T_j in terms of the set of processors $S \subseteq M$ that is assigned to T_j . Given a set S_j of processors assigned to task T_j , the processors in S_j are required to execute task T_j simultaneously; i.e., they all have to start processing task T_j at some starting time τ_j , and complete it at $\tau_j + t_j(S_j)$ or interrupt it before. A feasible preemptive schedule can be defined as a system of interval sequences $\{I_{j1} = [a_{j1}, b_{j1}), \dots, I_{js_j} = [a_{js_j}, b_{js_j})\}$, corresponding to the uninterrupted processing phases of task T_j , $j = 0, \dots, n - 1$, where $b_{jq} \leq a_{j,q+1}$ for every j and q . In addition there is an associated set $S_{jq} \subseteq M$ of processors for each interval I_{jq} such that $S_{jq} \neq S_{j,q+1}$ for every j and q (if $b_{jq} = a_{j,q+1}$). Moreover, for each time step τ there is no processor assigned to more than one task; i.e., $S_{jq} \cap S_{j'q'} = \emptyset$ for every τ such that $\tau \in I_{jq} \cap I_{j'q'}$, $j \neq j'$. The objective is to compute a feasible preemptive schedule that minimizes the overall makespan $C_{max} = \max\{b_{js_j} : j = 0, \dots, n - 1\}$. In section 2.1, we formulate this problem as a linear program and then, based on this linear programming formulation, we give an approximation scheme for $Pm|pmtn, set_j|C_{max}$ in section 2.2 with running time $O(n)$ for any fixed $\epsilon > 0$. Interestingly, the running time is polynomial in $\frac{1}{\epsilon}$.

2.1. Linear programming formulation. Let $\mathcal{L} \subseteq \mathcal{T}$ and $k = |\mathcal{L}|$. Later \mathcal{L} will be selected such that for any fixed m and $\epsilon > 0$, it consists of a constant number $k = \Theta(2^m(m/\epsilon))$ of “long” tasks. (For the choice of k we refer to section 2.4.) Let $P_1 = M, P_2 = M \setminus \{1\}, P_3 = M \setminus \{2\}, \dots, P_{2^m-1} = \{m\}, P_{2^m} = \emptyset$ be a fixed order of all subsets of M . Since preemptions are allowed, every feasible (and therefore also every optimal) schedule can be modified such that it remains feasible with the same makespan T and in addition has the following property: The time interval $[0, T)$ can be partitioned into 2^m consecutive subintervals $I(P_1) = [t_0 = 0, t_1), I(P_2) = [t_1, t_2), \dots, I(P_{2^m}) = [t_{2^m-1}, t_{2^m} = T)$ such that during $I(P_i), i = 1, \dots, 2^m$, tasks in \mathcal{L} are processed by *exactly* the processors in $P_i \subseteq M$. Each processor in P_i is working on one of the tasks from \mathcal{L} , and all the other processors are either idle or executing tasks from $\mathcal{S} = \mathcal{T} \setminus \mathcal{L}$. (For illustration see Figure 1, but notice that in contrast to the figure the lengths $t_{i+1} - t_i$ of the intervals are in general different and determined in the linear program below.)

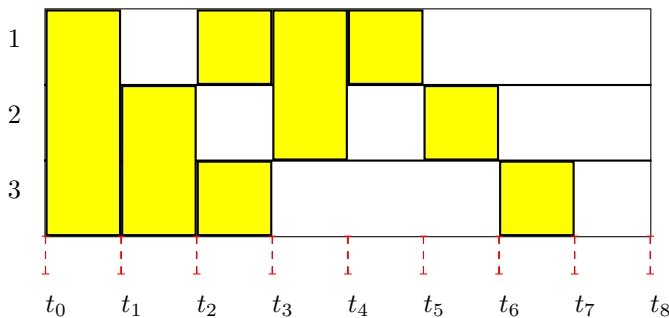


FIG. 1. Schedule with processors 1, 2, 3.

For tasks from \mathcal{L} and processors in $P_i, i = 1, \dots, 2^m - 1$, we consider assignments $f_i : P_i \rightarrow \mathcal{L}$. Let A_i denote the set containing all such assignments. Let $F_i = M \setminus P_i$ be the set of free processors for executing tasks from \mathcal{S} during the interval $I(P_i)$. Notice that $F_1 = \emptyset$ and $F_{2^m} = M$. For processor sets $F_i, i = 2, \dots, 2^m$, let $P_{F_i,q}$,

$q = 1, \dots, n_i$, denote the different partitions of F_i , and let $\mathcal{P}_i = \{P_{F_i,1}, \dots, P_{F_i,n_i}\}$. Finally, to simplify the notation below, we introduce the following indicators for every $\mu \subseteq M$ and every $P_{F_i,q} \in \mathcal{P}_i$, $i = 2, \dots, 2^m$: $\xi_{F_i,q}(\mu) = 1$ if $\mu \in P_{F_i,q}$ and 0 otherwise.

The following variables will be used in the linear program:

- t_i : the starting (and finishing) time of the interval when exactly the processors in P_{i+1} , $i = 0, \dots, 2^m - 1$ (P_i , $i = 1, \dots, 2^m$) are used for processing long tasks from \mathcal{L} .
- u_f : the total processing time corresponding to the assignment $f \in A_i$, $i = 1, \dots, 2^m - 1$.
- z_{ji} : the fraction of task $T_j \in \mathcal{L}$ processed in the interval $I(P_i) = [t_{i-1}, t_i]$, $i = 1, \dots, 2^m - 1$.
- $x_{F_i,q}$: the total processing time for partition $P_{F_i,q} \in \mathcal{P}_i$, $q = 1, \dots, n_i$, $i = 2, \dots, 2^m$, where exactly processors of F_i are executing short tasks and each subset of processors $\mu \in P_{F_i,q}$ executes at most one short task at each time step in parallel.
- D^μ : the total processing time for all tasks in $\mathcal{S} = \mathcal{T} \setminus \mathcal{L}$ executed on processor set μ .
- $y_{j\mu}$: the fraction of task $T_j \in \mathcal{S}$ executed by the processor set $\mu \subseteq M$, $\mu \neq \emptyset$. (By introducing these variables we can handle the possibility of nonfixed processor assignments also for the tasks in \mathcal{S} . Later, for the nonpreemptive version of the problem, the $y_{j\mu}$'s will be 0 – 1 variables.)

We consider the following linear program (which will be explained in detail following its complete statement).

Minimize t_{2^m}

- s.t. (0) $t_0 = 0$,
- (1) $t_i \geq t_{i-1}$, $i = 1, \dots, 2^m$,
 - (2) $t_i - t_{i-1} = \sum_{f \in A_i} u_f$, $i = 1, \dots, 2^m - 1$,
 - (3) $u_f \geq 0 \forall f \in A_i$, $i = 1, \dots, 2^m - 1$,
 - (4) $\sum_{\mu \subseteq P_i} \sum_{f \in A_i: f^{-1}(j)=\mu} \frac{u_f}{t_j(\mu)} = z_{ji} \forall j \in \mathcal{L}$, $i = 1, \dots, 2^m - 1$,
 - (5) $\sum_{i=1}^{2^m-1} z_{ji} = 1 \forall j \in \mathcal{L}$,
 - (6) $z_{ji} \geq 0 \forall j \in \mathcal{L}$, $i = 1, \dots, 2^m - 1$,
 - (7) $\sum_{q=1}^{n_i} x_{F_i,q} \leq t_i - t_{i-1}$, $i = 2, \dots, 2^m$,
 - (8) $\sum_{i=2}^{2^m} \sum_{q=1}^{n_i} \xi_{F_i,q}(\mu) \cdot x_{F_i,q} \geq D^\mu \forall \mu \subseteq M$, $\mu \neq \emptyset$,
 - (9) $x_{F_i,q} \geq 0$, $i = 2, \dots, 2^m$, $q = 1, \dots, n_i$,
 - (10) $\sum_{T_j \in \mathcal{S}} t_j(\mu) \cdot y_{j\mu} = D^\mu \forall \mu \subseteq M$, $\mu \neq \emptyset$,
 - (11) $\sum_{\mu \subseteq M, \mu \neq \emptyset} y_{j\mu} = 1 \forall T_j \in \mathcal{S}$,
 - (12) $y_{j\mu} \geq 0 \forall T_j \in \mathcal{S}$, $\forall \mu \subseteq M$, $\mu \neq \emptyset$.

Constraints (0)–(1) define the endpoints of the intervals $[t_i, t_{i+1})$, $i = 0, 1, \dots, 2^m - 1$. The further subdivision of these intervals corresponding to the different assignments in A_i is described by (2)–(3). The processing time of each long job can also be partitioned according to the processor sets P_i , $i = 1, \dots, 2^m - 1$, and each of these fractions has to be covered as it is formulated by (4)–(6). The inequalities of (7) require for every set of free processors F_i , $i = 2, \dots, 2^m$, that its total processing time (which is the sum of processing times corresponding to the different partitions) be bounded by the length of the interval $[t_{i-1}, t_i)$. Furthermore, the inequalities of (8) guarantee that there is enough time for the execution of all μ -processor tasks in \mathcal{S} . In (10), the total processing times D^μ required by short tasks using processor set $\mu \subseteq M$, $\mu \neq \emptyset$, are expressed in terms of variables $y_{j\mu}$. Finally, constraints (11) and (12) formulate the possibility of nonfixed processor assignments for short tasks in \mathcal{S} . Notice that

solutions of the above linear program allow for each task from \mathcal{S} to be executed parallel on different subsets μ of processors. Thus the schedule obtained directly from a solution of this linear program might contain some incorrectly scheduled tasks, which therefore have to be corrected afterwards. (See section 2.3.)

LEMMA 2.1. *In the preemptive schedule corresponding to any feasible solution of (0)–(12), every task T_j is processed completely.*

Proof. For every task $T_j \in \mathcal{T}$, let $w_j(\mu)$ be the total time while task T_j is processed by exactly the processors in $\mu \subseteq M$. Clearly task T_j is processed completely if and only if $\sum_{\mu \subseteq M} \frac{w_j(\mu)}{t_j(\mu)} \geq 1$. Since by definition $y_{j\mu} = \frac{w_j(\mu)}{t_j(\mu)}$ for $T_j \in \mathcal{S}$, the lemma is implied by (11) for the short tasks. For a long task $T_j \in \mathcal{L}$, $w_j(\mu) = \sum_{i=1}^{2^m-1} \sum_{f \in A_i: f^{-1}(j)=\mu} u_f$. Therefore we obtain by using (4) and (5) that

$$\begin{aligned} \sum_{\mu \subseteq M} \frac{w_j(\mu)}{t_j(\mu)} &= \sum_{\mu \subseteq M} \sum_{i=1}^{2^m-1} \sum_{f \in A_i: f^{-1}(j)=\mu} \frac{u_f}{t_j(\mu)} \\ &= \sum_{i=1}^{2^m-1} \sum_{\mu \subseteq M} \sum_{f \in A_i: f^{-1}(j)=\mu} \frac{u_f}{t_j(\mu)} = \sum_{i=1}^{2^m-1} z_{ji} = 1, \end{aligned}$$

which proves the lemma also for tasks from \mathcal{L} . \square

Let $d_j = \min_{\mu \subseteq M} t_j(\mu)$ be the minimum execution time for task T_j , and let $D = \sum_{T_j \in \mathcal{T}} d_j$. Then, the minimum makespan OPT among all schedules satisfies $\frac{D}{m} \leq OPT \leq D$. By normalization (dividing all execution times by D), we may assume that $D = 1$ and that $\frac{1}{m} \leq OPT \leq 1$. This implies that for any optimal schedule, and for every $\mu \subseteq M$, $\mu \neq \emptyset$, the total execution time D^μ of tasks executed by processor set μ can be bounded by 1.

2.2. Solving the linear program. Fix the length of the schedule to a constant $s \in [\frac{1}{m}, 1]$. Let $LP(s, \lambda)$ denote the linear program obtained from (0)–(12) by setting $t_{2^m} = s$ and replacing (8) and (10) with the following constraints:

$$(8') \quad \sum_{i=2}^{2^m} \sum_{q=1}^{n_i} \xi_{F_{i,q}}(\mu) \cdot x_{F_{i,q}} \leq 1, \quad \mu \subseteq M, \mu \neq \emptyset.$$

$$(10') \quad \sum_{T_j \in \mathcal{S}} t_j(\mu) \cdot y_{j\mu} - \sum_{i=2}^{2^m} \sum_{q=1}^{n_i} \xi_{F_{i,q}}(\mu) \cdot x_{F_{i,q}} + 1 \leq \lambda, \quad \mu \subseteq M, \mu \neq \emptyset.$$

Note that we have eliminated the variables D^μ but bounded the corresponding lengths for the configurations by 1. Since $OPT \leq 1$, we have $\sum_{i=2}^{2^m} \sum_{q=1}^{n_i} x_{F_{i,q}} \leq 1$ and $\sum_{i=2}^{2^m} \sum_{q=1}^{n_i} \xi_{F_{i,q}}(\mu) \cdot x_{F_{i,q}} \leq 1$ (for any optimal solution). The problem $LP(s, \lambda)$ has a special block angular structure. The blocks $B_j = \{y_j : y_{j\mu} \geq 0, \sum_{\mu \subseteq M} y_{j\mu} = 1\}$ for $T_j \in \mathcal{S}$ are 2^m -dimensional simplices, and the block $B_{|\mathcal{S}|+1} = \{(t_i, u_f, z_{ji}, x_{F_{i,q}}) : t_{2^m} = s \text{ and conditions (0)–(7), (8'), (9)}\}$ contains only a constant number of variables and constraints (the numbers depend exponentially on m , but are polynomial in $1/\epsilon$). The coupling constraints are the replaced linear inequalities (10'). Note that for every $\mu \subseteq M$, $\mu \neq \emptyset$, the function $f^\mu = \sum_{T_j \in \mathcal{S}} t_j(\mu) \cdot y_{j\mu} - \sum_{i=2}^{2^m} \sum_{q=1}^{n_i} \xi_{F_{i,q}}(\mu) \cdot x_{F_{i,q}} + 1$ is nonnegative over the blocks.

The logarithmic potential price directive decomposition method [10] developed by Grigoriadis and Khachiyan for a large class of problems with block angular structure provides a δ -relaxed decision procedure for $LP(s, \lambda)$. This procedure either determines that $LP(s, 1)$ is infeasible or computes (a solution that is nearly feasible in the sense that it is) a feasible solution of $LP(s, (1 + \delta))$. This can be done [10, 14] in $O(2^m(m + \delta^{-2}))$ iterations, where each iteration requires $O(2^m \ln \ln(2^m \delta^{-1}))$ operations and $|\mathcal{S}| + 1 \leq n + 1$ block optimizations performed to a relative accuracy of $\Theta(\delta)$, where $\delta =$

$\Theta(\epsilon)$. In our case each block optimization over $B_1, \dots, B_{|\mathcal{S}|}$ is the minimization of a given linear function over a 2^m -dimensional simplex which can be done (not only approximately, but even) exactly in $O(2^m)$ time. Furthermore, the block optimization over $B_{|\mathcal{S}|+1}$ is the minimization of a linear function over a block with a constant number of variables and constraints, which can be done in constant time. The number M of constraints in the last block is $O(2^m k)$, and the number N of variables is $O((2k)^m)$, where $k = O(2^m(m/\epsilon))$. The execution times $t_j(\mu)$ for tasks $T_j \in \mathcal{L}$ have values between $\epsilon/(2m2^{m+1})$ and 1 (see also section 2.4). Therefore we may assume that the coefficients in the inequalities (4) have values between 1 and $2^{O(m)}/\epsilon$. A linear program with N variables and M constraints can be solved in $O((MN^2 + M^{1.5}N)L)$ time for $M \geq N$, where L is the size of instance [18]. In our case $N > M$, and the running time can be bounded by $O(N^3L)$. The size of each coefficient (either in the inequalities or the linear objective function) is at most $O(m + \log(1/\epsilon))$. The size L can be bounded by $O(NM + (k2^m)(m + \log(1/\epsilon)))$, since there are at most $O(k2^m)$ different execution times $t_j(\mu)$ for tasks $T_j \in \mathcal{L}$. Therefore the running time to solve the linear program for block $B_{|\mathcal{S}|+1}$ is at most $O(N^5(k2^m)(m + \log 1/\epsilon)) = 2^{O(m^2)}(1/\epsilon)^{O(m)}$. This implies a running time for $LP(s, \lambda)$ of at most $[2^{O(m)}n\epsilon^{-2} + 2^{O(m^2)}(1/\epsilon)^{O(m)}]$. Therefore for any fixed m and $\delta > 0$, the overall running time of this procedure for (approximately) solving $LP(s, \lambda)$ is $O(n)$. The number of strictly positive variables corresponding to block $B_{|\mathcal{S}|+1}$ is at most $O(k2^{2m}(m + \epsilon^{-2})) = [2^{O(m)}\epsilon^{-3}]$.

The next lemma summarizes a few simple observations that can be easily checked.

LEMMA 2.2. *The following assertions are true:*

1. *If $LP(s, 1)$ is feasible and $s \leq s'$, then $LP(s', 1)$ is feasible.*
2. *If $LP(s, 1)$ is infeasible, then there is no schedule with makespan at most s .*
3. *$LP(OPT, 1)$ is feasible.*

These statements imply that one can use binary search on $s \in [\frac{1}{m}, 1]$ and obtain in $O(\log \frac{m}{\epsilon})$ iterations a value $\bar{s} \leq OPT(1 + \frac{\epsilon}{4})$ such that $LP(\bar{s}, (1 + \delta))$ is feasible.

2.3. Generating a schedule. In this subsection, we show how to generate a feasible schedule using an approximate solution of the previously described linear program. Consider the solution obtained after the binary search on s . The inequalities (10') imply that for any $\mu \subseteq M$, $\mu \neq \emptyset$,

$$\sum_{T_j \in \mathcal{S}} t_j(\mu) \cdot y_{j\mu} - \sum_{i=2}^{2^m} \sum_{q=1}^{n_i} \xi_{F_i,q}(\mu) \cdot x_{F_i,q} \leq \delta.$$

Let $\bar{D}^\mu = \sum_{i=2}^{2^m} \sum_{q=1}^{n_i} \xi_{F_i,q}(\mu) \cdot x_{F_i,q}$, the free space for small tasks that use processor set μ . The idea is to shift a subset $\bar{\mathcal{S}} \subset \mathcal{S}$ of small jobs to the end of the schedule such that $\sum_{T_j \in \mathcal{S} \setminus \bar{\mathcal{S}}} t_j(\mu) \cdot y_{j\mu} \leq \bar{D}^\mu$. Then, the subset $\mathcal{S} \setminus \bar{\mathcal{S}}$ of remaining small jobs fits into the free space for the μ -processor tasks.

In the following, we show how to compute such a subset $\bar{\mathcal{S}}$ in linear time for any fixed m . First we modify the y -components of the solution obtained from the linear program. The key idea is to allow no change of processor sets $\mu \subseteq M$ (or to have fractional y -components) for small tasks. Using this assumption it is easier to compute a set $\bar{\mathcal{S}}$ and a feasible schedule for the small tasks. Furthermore, only a few small tasks will have fractional y -components. The y -components of the approximate solution of $LP(\bar{s}, (1 + \delta))$ can be considered as fractional assignments. Let the lengths of y be defined as $L^\mu = \sum_{T_j \in \mathcal{S}} t_j(\mu) \cdot y_{j\mu}$, $\mu \subseteq M$. Then for every $\mu \subseteq M$, we have $L^\mu \leq \bar{D}^\mu + \delta$. A fractional assignment y can be represented by a bipartite graph

$G = (V_1, V_2, E)$, where V_1 and V_2 correspond to row and column indices of y (tasks and subsets of processors), respectively, and $(j, \mu) \in E$ if and only if $y_{j\mu} > 0$.

Any assignment y represented by a bipartite graph G of lengths L^μ , $\mu \subseteq M$, can be converted in $O(n2^{2m})$ time into another (fractional) assignment \tilde{y} of lengths at most L^μ , $\mu \subseteq M$, represented by a forest [17] (see Lemma 5.1). For a fractional assignment y , a task T_j has a nonunique processor assignment if there are at least two processor sets μ and μ' , $\mu \neq \mu'$, such that $y_{j\mu} > 0$ and $y_{j\mu'} > 0$. Let \mathcal{U}_1 be the set of tasks with nonunique processor assignments. For an assignment y corresponding to a forest, we have $|\mathcal{U}_1| \leq 2^m - 1$.

In the next step, we compute for every $\mu \subseteq M$ a subset $S^\mu \subset S \setminus \mathcal{U}_1$ of tasks with $y_{j\mu} = 1$ for all tasks $T_j \in S^\mu$ such that the total execution length $\sum_{T_j \in S^\mu} t_j(\mu) \geq \delta$ and there is a task $T_{j_\mu} \in S^\mu$ for which $\sum_{T_j \in S^\mu \setminus \{T_{j_\mu}\}} t_j(\mu) < \delta$. Let $\mathcal{U}_2 = \{T_{j_\mu} | \mu \subseteq M\}$.

In total, we get a set $\mathcal{U}_1 \cup \mathcal{U}_2$ with at most $2^{m+1} - 1$ tasks and a subset of tasks $\mathcal{V} = (\cup_{\mu \subseteq M} S^\mu) \setminus \mathcal{U}_2$ of total execution length $2^m \delta$. By choosing $\delta = \frac{\epsilon}{4m2^m}$, the total execution length of \mathcal{V} can be bounded by $\frac{\epsilon}{4m} \leq \frac{\epsilon}{4} OPT$. Using that $\bar{s} \leq (1 + \frac{\epsilon}{4}) OPT$, this implies the following lemma.

LEMMA 2.3. *The objective function value of the computed linear programming solution restricted to $\mathcal{T}' = \mathcal{T} \setminus (\mathcal{U}_1 \cup \mathcal{U}_2)$ is at most $OPT + \frac{\epsilon}{2} OPT$, and $|\mathcal{U}_1 \cup \mathcal{U}_2| \leq 2^{m+1} - 1$.*

The next step of the algorithm computes a schedule for the tasks in $\mathcal{T}'' = \mathcal{T} \setminus (\mathcal{U}_1 \cup \mathcal{U}_2 \cup \mathcal{V})$. (The tasks in $\mathcal{U}_1 \cup \mathcal{U}_2 \cup \mathcal{V}$ will be scheduled afterwards at the end.) The makespan of the computed schedule is bounded by $(1 + \frac{\epsilon}{4}) OPT$. Furthermore, the total execution time for \mathcal{V} is bounded by $\frac{\epsilon}{4} OPT$. Note that each small task $T_j \in \mathcal{T}' \cap \mathcal{S}$ has a unique set of assigned processors.

Let \hat{D}^μ be the total processing time for all tasks in $\mathcal{T}'' \cap \mathcal{S}$ assigned to processor set μ , and let $(t^*, u^*, z^*, x^*, y^*)$ be a solution of the linear program with objective value $t_{2^m}^*$. In the first step, we compute a schedule for the long tasks $T_j \in \mathcal{L}$ according to the variables t_i^* , u_i^* , and z_i^* . As a result, during the interval $[t_{i-1}^*, t_i^*)$ only the processors in P_i are used by the long tasks for every $i = 1, \dots, 2^m$. This first step can be done in time $[2^{O(m)} \epsilon^{-3}]$.

In the second step, we schedule all μ -processor tasks in $\mathcal{S}' = \mathcal{T}'' \cap \mathcal{S}$ for every $\mu \subseteq M$. From the left to the right (starting with the second interval), we place the tasks of \mathcal{S}' on the free processors in $F_i = M \setminus P_i$ for each interval $[t_{i-1}^*, t_i^*)$ (and $2 \leq i \leq 2^m$). To do this, we consider each partition $P_{F_i, q}$ of F_i with value $x_{F_i, q}^* > 0$. For each set $\mu \in P_{F_i, q}$, we place a sequence of tasks that use processor set μ with total execution length $x_{F_i, q}^*$. If necessary, the last (and first) task assigned to μ is preempted. Since $\sum_{i=2}^{2^m} \sum_{q=1}^{n_i} \xi_{F_i, q}(\mu) \cdot x_{F_i, q}^* \geq \hat{D}^\mu$, this procedure completely schedules all small tasks (assigned to processor set μ for every $\mu \subseteq M$), and it runs in $[O(n) + 2^{O(m)} \epsilon^{-3}]$ time. Note that the computed schedule is feasible.

2.4. Selecting the cardinality of \mathcal{L} . The following lemma will be used to select the cardinality k of \mathcal{L} (a constant for any fixed m and $\epsilon > 0$), such that the total processing time of $\mathcal{U}_1 \cup \mathcal{U}_2$ can be bounded by $\frac{\epsilon}{2} OPT$. Note that a more general version of this lemma was proved in [12].

LEMMA 2.4. *Suppose $d_1 \geq d_2 \geq \dots \geq d_n > 0$ is a sequence of real numbers and $D = \sum_{j=1}^n d_j$. Let p be a nonnegative integer, $\alpha \in (0, 1)$, and assume that n is sufficiently large (i.e., all the indices of the d_i 's in the statement are smaller than n ; e.g., $n > (\lceil \frac{1}{\alpha} \rceil p + 1)$ suffices). Then, there exists an integer $k = k(p, \alpha)$ such that*

$$d_k + \dots + d_{k+p-1} \leq \alpha \cdot D$$

and

$$k \leq 1 + p \left(\left\lceil \frac{1}{\alpha} \right\rceil - 1 \right).$$

In our problem $D = \sum_{T_j \in \mathcal{T}} d_j$, where $d_j = \min_{\mu \subseteq M} t_j(\mu)$ is the minimum execution time for task T_j . The minimum makespan OPT among all schedules satisfies $\frac{D}{m} \leq OPT \leq D$. By normalization, we assumed that $D = 1$ and that $\frac{1}{m} \leq OPT \leq 1$. We select now $p = 2^{m+1} - 1$ and $\alpha = \frac{\epsilon}{2^m}$ in Lemma 2.4. Then, Lemmas 2.3 and 2.4 imply that there exists a constant $k \leq 1 + (2^{m+1} - 1)(\lceil \frac{2^m}{\epsilon} \rceil - 1)$ such that the total execution time of the tasks in $\mathcal{U}_1 \cup \mathcal{U}_2$ can be bounded by $\frac{\epsilon}{2}OPT$. Furthermore, the makespan for the partial (feasible) schedule of $\mathcal{T} \setminus (\mathcal{U}_1 \cup \mathcal{U}_2)$ is at most $(1 + \frac{\epsilon}{2})OPT$. Thus the overall makespan of the (complete) schedule is bounded by $(1 + \epsilon)OPT$. According to the arguments above all computations can be carried out in $[2^{O(m)}(1/\epsilon)^2n + 2^{O(m^2)}(1/\epsilon)^m]$ time, which implies a running time of $O(n)$ for any fixed m and $\epsilon > 0$. Thus, we have proved the following result.

THEOREM 2.5. *There is an algorithm that, given a set of n independent tasks, a constant number of processors m , a fixed positive accuracy ϵ , and execution times $t_j(\mu)$ for each task T_j and subset of processors μ , produces a preemptive schedule whose makespan is at most $(1 + \epsilon)OPT$ in $O(n)$ time.*

Clearly for any fixed m , the running time of the above algorithm depends only polynomially on $\frac{1}{\epsilon}$; hence it is a fully polynomial approximation scheme.

3. Nonpreemptive scheduling. In this section, we study the nonpreemptive variant $Pm|set_j|C_{max}$ of the general multiprocessor scheduling problem. Given a set S_j of processors assigned to task T_j , the processors in S_j are required to execute task T_j in union and without preemption; i.e., they all have to start processing task T_j at some starting time τ_j and complete it at $\tau_j + t_j(S_j)$. A feasible nonpreemptive schedule consists of a processor assignment $S_j \subseteq M$ and a starting time $\tau_j \geq 0$ for each task T_j such that for each time step τ , there is no processor assigned to more than one task. The objective is to find a feasible nonpreemptive schedule that minimizes the overall makespan $C_{max} = \max\{\tau_j + t_j(S_j) : j = 0, \dots, n - 1\}$.

3.1. Linear programming formulation. First we consider a mixed 0 – 1 integer linear program which is closely related to our scheduling problem. Similar formulations were studied in [1, 12] for restricted (dedicated and malleable) variants of the problem. Based on this linear programming formulation we will give a linear-time approximation scheme for $Pm|set_j|C_{max}$.

Let $\mathcal{L} \subset \mathcal{T}$. A processor assignment for \mathcal{L} is a mapping $f : \mathcal{L} \rightarrow 2^M$. Two tasks T_j and $T_{j'}$ are *compatible* if $f(T_j) \cap f(T_{j'}) = \emptyset$. For a given processor assignment for \mathcal{L} , a *snapshot* of \mathcal{L} is a subset of compatible tasks. A *relative schedule* of \mathcal{L} is a processor assignment $f : \mathcal{L} \rightarrow 2^M$ along with a sequence $M(1), \dots, M(g)$ of snapshots of \mathcal{L} such that

- each $T_j \in \mathcal{L}$ occurs in a subset of consecutive snapshots $M(\alpha_j), \dots, M(\omega_j)$, $1 \leq \alpha_j \leq \omega_j < g$, where $M(\alpha_j)$ is the first and $M(\omega_j)$ is the last snapshot that contains T_j ;
- consecutive snapshots are different; i.e., $M(t) \neq M(t + 1)$ for $1 \leq t \leq g - 1$;
- $M(1) \neq \emptyset$ and $M(g) = \emptyset$.

A relative schedule corresponds to an order of executing the tasks in \mathcal{L} . One can associate a relative schedule for each nonpreemptive schedule of \mathcal{L} by looking at the schedule at every time where a task of \mathcal{L} starts or ends and creating a snapshot

right after that time step. Creating snapshots this way, $M(1) \neq \emptyset$, $M(g) = \emptyset$, and g , the number of snapshots, can be bounded by $\max(2|\mathcal{L}|, 1)$. Given a relative schedule $R = (f, M(1), \dots, M(g))$, the processor set used in snapshot $M(i)$ is given by $P(i) = \cup_{T \in M(i)} f(T)$. Let \mathcal{F} denote the set containing (as elements) all the different $(M \setminus P(i))$ sets, $i = 1, \dots, g$. (Thus the sets in \mathcal{F} are the different sets of free processors corresponding to R .) For each $F \in \mathcal{F}$, let $P_{F,i}, i = 1, \dots, n_F$, denote the different partitions of F , and let $\mathcal{P}_F = \{P_{F,1}, \dots, P_{F,n_F}\}$. Furthermore, let D^μ be the total processing time for all tasks in $\mathcal{S} = \mathcal{T} \setminus \mathcal{L}$ executed on processor set $\mu \subseteq M$. Finally, we define the indicators for every $\mu \subseteq M$ and every $P_{F,i} \in \mathcal{P}_F, i = 1, \dots, n_F, F \in \mathcal{F}$, similarly as before: $\xi_{F,i}(\mu) = 1$ if $\mu \in P_{F,i}$ and 0 otherwise.

For each relative schedule $R = (f, M(1), \dots, M(g))$ of \mathcal{L} , we formulate a mixed 0–1 integer program $ILP(R)$ as follows (where again we defer a complete explanation of the linear program until after its complete statement). For every $T_j \in \mathcal{L}$, let $p_j = t_j(f(T_j))$.

Minimize t_g

- s.t. (0) $t_0 = 0$,
- (1) $t_i \geq t_{i-1}, i = 1, \dots, g$,
 - (2) $t_{\omega_j} - t_{\alpha_j-1} = p_j \forall T_j \in \mathcal{L}$,
 - (3) $\sum_{i:P(i)=M \setminus F} (t_i - t_{i-1}) = e_F \forall F \in \mathcal{F}$,
 - (4) $\sum_{i=1}^{n_F} x_{F,i} \leq e_F \forall F \in \mathcal{F}$,
 - (5) $\sum_{F \in \mathcal{F}} \sum_{i=1}^{n_F} \xi_{F,i}(\mu) \cdot x_{F,i} \geq D^\mu \forall \mu \subseteq M, \mu \neq \emptyset$,
 - (6) $x_{F,i} \geq 0 \forall F \in \mathcal{F}, i = 1, \dots, n_F$,
 - (7) $\sum_{T_j \in \mathcal{S}} t_j(\mu) \cdot y_{j\mu} = D^\mu \forall \mu \subseteq M, \mu \neq \emptyset$,
 - (8) $\sum_{\mu \subseteq M} y_{j\mu} = 1 \forall T_j \in \mathcal{S}$,
 - (9) $y_{j\mu} \in \{0, 1\} \forall T_j \in \mathcal{S}, \forall \mu \subseteq M, \mu \neq \emptyset$.

The variables of $ILP(R)$ have the following interpretation:

t_i : the time when snapshot $M(i)$ ends (and $M(i + 1)$ starts), $i = 1, \dots, g - 1$.

The starting time of the schedule and snapshot $M(1)$ is denoted by $t_0 = 0$ and the finishing time by t_g .

e_F : the total time while exactly the processors in F are free.

$x_{F,i}$: the total processing time for $P_{F,i} \in \mathcal{P}_F, i = 1, \dots, n_F, F \in \mathcal{F}$, where only processors of F are executing short tasks and each subset of processors $F_j \in P_{F,i}$ executes at most one short task at each time step in parallel.

$y_{j\mu}$: the assignment variable indicating whether task $T_j \in \mathcal{S}$ is executed on processor set μ , i.e.,

$$y_{j\mu} = \begin{cases} 1 & \text{if } T_j \text{ is executed by the processor set } \mu, \\ 0 & \text{otherwise.} \end{cases}$$

The given relative schedule R along with constraints (1) and (2) define a feasible schedule of \mathcal{L} . In (3), the total processing times e_F for all $F \in \mathcal{F}$ are determined. Clearly, these equalities can be inserted directly into (4). The inequalities in (4) require for every set of free processors $F \in \mathcal{F}$ that its total processing time (corresponding to the different partitions) be bounded by e_F . Furthermore, the inequalities (5) guarantee that there is enough time for the execution of all tasks in \mathcal{S} that use processor set μ . The constraints (8)–(9) describe the processor assignments for the tasks in \mathcal{S} . The equations of (7) express for every $\mu \subseteq M, \mu \neq \emptyset$, the total processing time D^μ of all tasks in \mathcal{S} that are executed by the processor set μ . As before, these equations can be inserted directly into the inequalities of (5).

We will use the relaxation $LP(R)$ of $ILP(R)$, where we replace the 0–1 constraints by the inequalities $y_{j\mu} \geq 0$ for every $T_j \in \mathcal{S}$ and $\mu \subseteq M$. Notice that the solutions of $LP(R)$ allow for each task from \mathcal{S} to be preempted or to be executed on different subsets μ of processors. Thus there might be incorrectly scheduled tasks in the schedule based on the solution of $LP(R)$. These have to be corrected afterwards. It is easy to check that the approach we used in section 2.2 for solving the linear program of the preemptive problem can also be applied to $LP(R)$. This way one can obtain the same approximation bounds as those in section 2.2.

3.2. Generating a schedule. Similarly to the preemptive variant, one can also compute a subset $\bar{\mathcal{S}} \subset \mathcal{S}$ of small jobs such that $\sum_{T_j \in \bar{\mathcal{S}}} t_j(\mu) \cdot y_{j\mu} \leq \bar{D}^\mu$; i.e., the set $\mathcal{S} \setminus \bar{\mathcal{S}}$ of remaining small tasks fits into the free space for the μ -processor tasks. The computation of $\bar{\mathcal{S}}$ can be done in the exact same manner as in section 2.3. Let \mathcal{U}_1 be the set of tasks with nonunique processor assignments, and let \mathcal{U}_2 be a subset of $\bar{\mathcal{S}}$ as described in section 2.3. Hence the following lemma also holds.

LEMMA 3.1. *The objective function value of the best approximate solution of $LP(R)$ (over all relative schedules R) restricted to $\mathcal{T}' = \mathcal{T} \setminus (\mathcal{U}_1 \cup \mathcal{U}_2)$ is at most $OPT + \frac{\epsilon}{2}OPT$, and $|\mathcal{U}_1 \cup \mathcal{U}_2| \leq 2^{m+1} - 1$.*

The next step of the algorithm requires the computation of a pseudoschedule $PS(\mathcal{T}'')$ for the tasks in $\mathcal{T}'' = \mathcal{T} \setminus (\mathcal{U}_1 \cup \mathcal{U}_2 \cup \mathcal{V})$, in which we allow that some tasks from \mathcal{S} are preempted. The makespan of the computed pseudoschedule is at most $OPT + \frac{\epsilon}{4}OPT$. Furthermore, the total execution time for \mathcal{V} is at most $\frac{\epsilon}{4}OPT$. We note that each task $T_j \in \mathcal{T}' \cap \mathcal{S}$ has a unique subset of assigned processors. Let \hat{D}^μ be the total processing time for all tasks in $\mathcal{T}'' \cap \mathcal{S}$ assigned to subset μ . We schedule all μ -processor tasks in $\mathcal{S}' = \mathcal{T}'' \cap \mathcal{S}$ for every subset $\mu \subseteq M$. From left to right (starting with the first snapshot $M(1)$), we place the tasks of \mathcal{S}' on the free processors in $M \setminus P(i)$ for each snapshot $M(i)$ (and $0 \leq i \leq g$). To do this, we consider each partition $P_{F(i),\ell}$ of $F(i) = M \setminus P(i)$ with value $x_{F(i),\ell}^* > 0$. For each set μ in the partition $P_{F(i),\ell}$, we place a sequence of tasks that use processor set μ with total execution length $x_{F(i),\ell}^*$. If necessary, the last (and first) task assigned to μ is preempted. Since $\sum_{F \in \mathcal{F}} \sum_{i=1}^{n_F} \xi_{F,i}(\mu) \cdot x_{F,i} \geq \hat{D}^\mu$, this procedure completely schedules all tasks (assigned to processor set μ) for every $\mu \subseteq M$, and it runs in $O(km^m + n) = [m^{O(m/\epsilon)} + O(n)]$ time (using $k = m^{O(m/\epsilon)}$; see Lemma 3.3), which leads to a running time of $O(n)$ for any fixed m and $\epsilon > 0$. Let \mathcal{W} be the set of preempted (and therefore incorrectly scheduled) tasks in $PS(\mathcal{T}'')$. The following lemma gives an upper bound on the cardinality of \mathcal{W} .

LEMMA 3.2. $|\mathcal{W}| \leq 2(m-1)k + 2m^{m+1}$.

Proof. First, we may assume that all partitions with m free processors are in the last interval $[t_{g-1}, t_g]$; otherwise we can shift all such intervals to the end of the schedule. There are two different cases of incorrectly scheduled tasks: 1. inside of an interval $[t_i, t_{i+1})$ caused by a change from a partition $P_{F,a}$ to $P_{F,a+1}$; 2. at the end of an interval $[t_i, t_{i+1})$ or inside of an interval without a change of partitions. In the second case, we have at most $(m-1)$ new incorrectly scheduled tasks (not counted before) for each interval $[t_i, t_{i+1})$. In the interval $[t_{g-1}, t_g)$, the last selected tasks are either not preempted or counted before. In total, we obtain at most $(m-1)(g-1) \leq 2(m-1)k$ incorrectly scheduled tasks in case 2. In the first case, for each change from a partition $P_{F,a}$ to $P_{F,a+1}$ we get at most m incorrectly scheduled tasks. The number of all partitions can be bounded by $m! + m^m \leq 2m^m$, which implies the bound $2m^{m+1}$ for incorrectly scheduled tasks in case 1. \square

Lemmas 3.1 and 3.2 imply the following upper bound (Corollary 3.3) on the total number of tasks that we have to shift to the end of the schedule.

COROLLARY 3.3. $|\mathcal{U}_1 \cup \mathcal{U}_2 \cup \mathcal{W}| \leq 2(m-1)k + 2m^{m+1} + 2^m \leq 2(m-1)k + 3m^{m+1}$.

The following general lemma which was proved in [13] will be used to select the cardinality k of \mathcal{L} (a constant for any fixed m and $\epsilon > 0$), such that the total processing time of $\mathcal{U}_1 \cup \mathcal{U}_2 \cup \mathcal{W}$ can be bounded. We mention in passing that this lemma (or various simplified variants and corollaries of it) can be used in designing polynomial approximation schemes for other scheduling problems as well; see [1, 7, 12, 13].

LEMMA 3.4. *Suppose $d_1 \geq d_2 \geq \dots \geq d_n > 0$ is a sequence of real numbers and $D = \sum_{j=1}^n d_j$. Let p, q be nonnegative integers, $\alpha > 0$, and assume that n is sufficiently large (i.e., all the indices of the d_i 's in the statement are smaller than n ; e.g., $n > (\lceil \frac{1}{\alpha} \rceil p + 1)(q + 1)^{\lceil \frac{1}{\alpha} \rceil}$ suffices). Then, there exists an integer $k = k(p, q, \alpha)$ such that*

$$d_k + \dots + d_{k+p+qk-1} \leq \alpha \cdot D$$

and

$$k \leq (q + 1)^{\lceil \frac{1}{\alpha} \rceil - 1} + p[1 + (q + 1) + \dots + (q + 1)^{\lceil \frac{1}{\alpha} \rceil - 2}].$$

If $q > 0$, the bound on k simplifies to $[(p + q)(q + 1)^{\lceil \frac{1}{\alpha} \rceil - 1} - p]/q$. By applying Lemma 3.4 with values $p = 3m^{m+1}$, $q = 2(m - 1)$, and $\alpha = \frac{\epsilon}{2m}$, we obtain that there exists a constant $k = k(m, \epsilon) \leq 1 + 4m^m(2m)^{2m/\epsilon} = m^{O(m/\epsilon)}$ such that the total execution time of the tasks in $\mathcal{U}_1 \cup \mathcal{U}_2 \cup \mathcal{W}$ can be bounded by $\frac{\epsilon}{2m} \leq \frac{\epsilon}{2}OPT$. Notice that $k = m^{O(m/\epsilon)}$ is exponential in $1/\epsilon$ and that the number of all relative schedules can be bounded by a constant $k^{O(k)}$. The overall running time can be bounded by $[k^{O(k)}(TIME(LP(R), \epsilon) + m^{O(m/\epsilon)} + n)]$, where $TIME(LP(R), \epsilon)$ denotes the time required to calculate an ϵ -approximate solution for the linear program $LP(R)$. Using the same arguments as in section 2.2 one can show that for any fixed m and $\epsilon > 0$, the linear program $LP(R)$ can be solved with ϵ -accuracy in $O(n)$ time. Furthermore, the makespan for the partial (feasible) schedule of $\mathcal{T} \setminus (\mathcal{U}_1 \cup \mathcal{U}_2)$ is at most $(1 + \frac{\epsilon}{2})OPT$. Thus, the overall makespan of the (complete) schedule is bounded by $(1 + \epsilon)OPT$. As stated by the arguments above, for any fixed m and $\epsilon > 0$, all computations can be carried out in $O(n)$ time. Therefore the following result holds for $Pm|set_j|C_{max}$.

THEOREM 3.5. *There is an algorithm which, given a set of n independent tasks, a constant number of processors m , a fixed positive accuracy ϵ , and execution times $t_j(\mu)$, for each task T_j and subset of processors μ , calculates in $O(n)$ time a nonpreemptive schedule whose makespan is at most $(1 + \epsilon)OPT$.*

4. Conclusions. In the present paper, we have proposed for the scheduling problem $Pm|pmtn, set_j|C_{max}$ a fully polynomial approximation scheme which for any fixed positive accuracy computes an approximate solution in $O(n)$ time. The previous approach [2] to this problem was based on a linear programming formulation with $O(n^m)$ variables, and hence it had a substantially larger time complexity in terms of n .

We have also studied the nonpreemptive version $Pm|set_j|C_{max}$ of the problem and presented a polynomial-time approximation scheme whose running time depends only linearly on n . Regarding the running time, this gives a significant improvement on the recent result obtained (independently from our work) by Chen and Miranda [7]. In addition, our approach avoids the use of dynamic programming and hence also answers an open question in [7].

REFERENCES

- [1] A. K. AMOURA, E. BAMPIS, C. KENYON, AND Y. MANOUSSAKIS, *Scheduling independent multiprocessor tasks*, *Algorithmica*, 32 (2001), pp. 247–261.
- [2] L. BIANCO, J. BLAZEWICZ, P. DELL OLMO, AND M. DROZDOWSKI, *Scheduling multiprocessor tasks on a dynamic configuration of dedicated processors*, *Ann. Oper. Res.*, 58 (1995), pp. 493–517.
- [3] J. BLAZEWICZ, P. DELL OLMO, M. DROZDOWSKI, AND M. SPERANZA, *Scheduling multiprocessor tasks on the three dedicated processors*, *Inform. Process. Lett.*, 41 (1992), pp. 275–280.
- [4] J. BLAZEWICZ, P. DELL OLMO, M. DROZDOWSKI, AND M. SPERANZA, *Corrigendum to: “Scheduling multiprocessor tasks on the three dedicated processors,”* *Inform. Process. Lett.*, 49 (1994), pp. 269–270.
- [5] J. BLAZEWICZ, M. DRABOWSKI, AND J. WEGLARZ, *Scheduling multiprocessor tasks to minimize schedule length*, *IEEE Trans. Comput.*, 35 (1986), pp. 389–393.
- [6] J. CHEN AND C.-Y. LEE, *General multiprocessor task scheduling*, *Naval Res. Logist.*, 46 (1999), pp. 57–74.
- [7] J. CHEN AND A. MIRANDA, *A polynomial time approximation scheme for general multiprocessor job scheduling*, *SIAM J. Comput.*, 31 (2001), pp. 1–17.
- [8] M. DROZDOWSKI, *Scheduling multiprocessor tasks—an overview*, *European J. Oper. Res.*, 94 (1996), pp. 215–230.
- [9] J. DU AND J. Y.-T. LEUNG, *Complexity of scheduling parallel task systems*, *SIAM J. Discrete Math.*, 2 (1989), pp. 473–487.
- [10] M. D. GRIGORIADIS AND L. G. KHACHIYAN, *Coordination complexity of parallel price-directive decomposition*, *Math. Oper. Res.*, 21 (1996), pp. 321–340.
- [11] J. A. HOOGEVEEN, S. L. VAN DE VELDE, AND B. VELTMAN, *Complexity of scheduling multiprocessor tasks with prespecified processor allocations*, *Discrete Appl. Math.*, 55 (1994), pp. 259–272.
- [12] K. JANSEN AND L. PORKOLAB, *Linear-time approximation schemes for scheduling malleable parallel tasks*, *Algorithmica*, 32 (2002), pp. 507–520.
- [13] K. JANSEN AND L. PORKOLAB, *Improved approximation schemes for scheduling unrelated parallel machines*, *Math. Oper. Res.*, 26 (2001), pp. 324–338.
- [14] K. JANSEN AND H. ZHANG, *Approximation algorithms for general packing problems with modified logarithmic potential function*, in *Proceedings of the 2nd IFIP Conference on Theoretical Computer Science, 2002*, pp. 255–266.
- [15] C. KENYON AND E. REMILA, *A near-optimal solution to a two-dimensional cutting stock problem*, *Math. Oper. Res.*, 25 (2000), pp. 645–656.
- [16] W. LUDWIG AND P. TIWARI, *Scheduling malleable and nonmalleable parallel tasks*, in *Proceedings of the 5th Annual ACM-SIAM Symposium on Discrete Algorithms*, ACM, New York, SIAM, Philadelphia, 1994, pp. 167–176.
- [17] S. A. PLOTKIN, D. B. SHMOYS, AND E. TARDOS, *Fast approximation algorithms for fractional packing and covering problems*, *Math. Oper. Res.*, 20 (1995), pp. 257–301.
- [18] P. M. VAIDYA, *An algorithm for linear programming which requires $O(((m+n)n^2 + (m+n)^{1.5}n)L)$ arithmetic operations*, *Math. Programming*, 47 (1990), pp. 175–201.

OPTIMAL COVERING TOURS WITH TURN COSTS*

ESTHER M. ARKIN[†], MICHAEL A. BENDER[‡], ERIK D. DEMAINE[§],
SÁNDOR P. FEKETE[¶], JOSEPH S. B. MITCHELL[†], AND SAURABH SETHIA^{||}

Abstract. We give the first algorithmic study of a class of “covering tour” problems related to the geometric traveling salesman problem: Find a polygonal tour for a *cutter* so that it sweeps out a specified region (“pocket”) in order to minimize a cost that depends mainly on the number of *turns*. These problems arise naturally in manufacturing applications of computational geometry to automatic tool path generation and automatic inspection systems, as well as arc routing (“postman”) problems with turn penalties. We prove the NP-completeness of minimum-turn milling and give efficient approximation algorithms for several natural versions of the problem, including a polynomial-time approximation scheme based on a novel adaptation of the *m*-guillotine method.

Key words. NC machining, manufacturing, traveling salesman problem, milling, lawn mowing, covering, approximation algorithms, polynomial-time approximation scheme, *m*-guillotine subdivisions, NP-completeness, turn costs

AMS subject classifications. 90C27, 68W25, 68Q25

DOI. 10.1137/S0097539703434267

1. Introduction. An important algorithmic problem in manufacturing is to compute effective paths and tours for covering (“milling”) a given region (“pocket”) with a cutting tool. The objective is to find a path or tour along which to move a prescribed cutter in order that the sweep of the cutter covers the region, removing all of the material from the pocket, while not “gouging” the material that lies outside of the pocket. This covering tour or “lawn mowing” problem [6] and its variants arise not only in numerically controlled (NC) machining applications but also in automatic inspection, spray painting/coating operations, robotic exploration, arc routing, and even mathematical origami.

The majority of research on these geometric covering tour problems as well as on the underlying arc routing problems in networks has focused on cost functions based on the lengths of edges. However, in many actual routing problems, this cost is dominated by the cost of switching paths or direction at a junction. A drastic example is given by

*Received by the editors September 10, 2003; accepted for publication (in revised form) July 13, 2005; published electronically December 8, 2005. An extended abstract version of this paper appeared in *Proceedings of the 12th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 01)*, 2001, pp. 138–147 [4]. This research was partially supported by the National Science Foundation (CCR-9732221, CCR-0098172) and by grants from Bridgeport Machines, HRL Laboratories, ISX Corporation, Metron Aviation, NASA (NAG2-1325), Northrop-Grumman, Sandia National Labs, Seagull Technology, and Sun Microsystems.

<http://www.siam.org/journals/sicomp/35-3/43426.html>

[†]Department of Applied Mathematics and Statistics, State University of New York, Stony Brook, NY 11794-3600 (estie@ams.sunysb.edu, jsbm@ams.sunysb.edu).

[‡]Department of Computer Science, State University of New York, Stony Brook, NY 11794-4400 (bender@cs.sunysb.edu).

[§]Computer Science and Artificial Intelligence Laboratory, Massachusetts Institute of Technology, Cambridge, MA 02139 (edemaine@mit.edu).

[¶]Department of Mathematical Optimization, Braunschweig University of Technology, Pockelsstr. 14, 38106 Braunschweig, Germany (s.fekete@tu-bs.de). Various parts of this work were done while this author visited Stony Brook, with partial support by DFG travel grants Fe407.

^{||}SoftJin Infotech Pvt. Ltd., Bangalore, India (saurabhsethia@gmail.com). This author participated in research while affiliated with the Department of Applied Mathematics and Statistics, SUNY Stony Brook.

fiber-optical networks, where the time to follow an edge is negligible compared to the cost of changing to a different frequency at a router. In the context of NC machining, turns represent an important component of the objective function, as the cutter may have to be slowed in anticipation of a turn. The number of turns (“link distance”) also arises naturally as an objective function in robotic exploration (minimum-link watchman tours) and in various arc routing problems, such as snow plowing or street sweeping with turn penalties. Klein [35] has posed the question of minimizing the number of turns in polygon exploration problems.

In this paper, we address the problem of minimizing the cost of *turns* in a covering tour. This important aspect of the problem has been left unexplored so far in the algorithmic community, and the arc routing community has examined only heuristics without performance guarantee, or exact algorithms with exponential running time. Thus, our study provides important new insights and a better understanding of the problems arising from turn cost. We present several new results:

- (1) We prove that the covering tour problem with turn costs is NP-complete, even if the objective is purely to minimize the number of turns, the pocket is orthogonal (rectilinear), and the cutter must move axis-parallel. The hardness of the problem is not apparent, as our problem seemingly bears a close resemblance to the polynomially solvable Chinese postman problem; see the discussion below.
- (2) We provide a variety of constant-factor approximation algorithms that efficiently compute covering tours that are nearly optimal with respect to turn costs in various versions of the problem. While getting *some* $O(1)$ -approximation is not difficult for most problems in this class, through a careful study of the structure of the problem, we have developed tools and techniques that enable significantly stronger approximation results.

One of our main results is a 3.75-approximation for minimum-turn axis-parallel tours for a unit square cutter that covers an integral orthogonal polygon, possibly with holes. Another main result gives a $4/3$ -approximation for minimum-turn tours in a “thin” pocket, as arises in the arc routing version of our problem.

Table 1.1 summarizes our results. The term “coverage” indicates the number of times a point is visited, which is of interest in several practical applications. This parameter also provides an upper bound on the total length.

- (3) We devise a polynomial-time approximation scheme (PTAS) for the covering tour problem in which the cost is given as a weighted combination of length and number of turns, i.e., the Euclidean length plus a constant C times the number of turns. For an integral orthogonal polygon with h holes and N pixels, the running time is $2^{O(h)} \cdot N^{O(C)}$. The PTAS involves an extension of the m -guillotine method, which has previously been applied to obtain PTASs in problems involving only *length* [38].

We should stress that our paper focuses on the graph-theoretic and algorithmic aspects of the turn-cost problem; we make no claims of immediate applicability of our methods to NC machining.

Related work. In the CAD community, there is a vast literature on the subject of automatic tool-path generation; we refer the reader to Held [27] for a survey and for applications of computational geometry to the problem. The algorithmic study of the problem has focused on the problem of minimizing the length of a milling tour: Arkin, Fekete, and Mitchell [5, 6] show that the problem is NP-hard for the case

TABLE 1.1

Approximation factors achieved by our (polynomial-time) algorithms. Rows marked “APX” give approximation factors for the minimum-turn cycle cover (“Cycle cover”), the minimum-turn covering tour (“Tour”), and the simultaneous approximation of length of a covering tour (“Length”). The row marked “Max cover” indicates the maximum number of times a point is visited. The parameter δ denotes the maximum degree in the underlying graph, while the parameter ρ is the maximum number of directions in the graph. The two rows for running time refer to an “explicit” description of input pixels and output and a more compact “implicit” encoding of pixels and output. (See section 2 for more detailed definitions.)

	Discrete milling	Thin discrete		Orthogonal milling	Thin orthogonal
Section	5.1.1	5.1.2		5.3	5.4
Cycle cover APX	$2\delta + \rho$	4	1.5	4.5	1
Tour APX	$2\delta + \rho + 2$	6	3.5	6.25	4/3
Length APX	δ	2	-	8	4
Max cover	δ	2ρ	-	8	4
Time (explicit)	$O(N)$	$O(N)$	$O(N^3)$	$O(N^{2.376} + n^3)$	$O(n^3)$
Time (implicit)	n/a	n/a	n/a	$O(n^{2.5} \log N + n^3)$	$O(n^3)$

	Integral orthogonal		
Section	5.2		
Cycle cover APX	10	4	2.5
Tour APX	12	6	3.75
Length APX	4	4	4
Max cover	4	4	4
Time (explicit)	$O(N)$	$O(N^{2.376})$	$O(N^{2.376} + n^3)$
Time (implicit)	$O(n \log n)$	$O(n^{2.5} \log N)$	$O(n^{2.5} \log N + n^3)$

where the mower is a square. Constant-factor approximation algorithms are given in [5, 6, 30], with the current best factor being a 2.5-approximation for min-length milling (11/5-approximation for orthogonal simple polygons). For the closely related *lawn mowing* problem (also known as the “traveling cameraman problem” [30]), in which the covering tour is not constrained to stay within P , the best current approximation factor is $3 + \epsilon$ (utilizing PTAS results for the traveling salesman problem (TSP)). Also closely related is the watchman route problem with limited visibility (or “ d -sweeper problem”); Ntafos [42] provides a 4/3-approximation, and Arkin, Fekete, and Mitchell [6] improve this factor to 6/5. The problem is also closely related to the Hamiltonicity problem in grid graphs; the results of [44] suggest that in *simple* polygons, minimum-length milling may in fact have a polynomial-time algorithm.

Covering tour problems are related to *watchman route* problems in polygons, which have received considerable attention in terms of both exact algorithms (for the simple polygon case) and approximation algorithms (in general); see [39] for a relatively recent survey. Most relevant to our problem is the prior work on *minimum-link* watchman tours: see [2, 3, 8] for hardness and approximation results, and [14, 36] for combinatorial bounds. However, in these problems the watchman is assumed to see arbitrarily far, making them distinct from our tour cover problems.

Other algorithmic results on milling include a study of *multiple tool* milling by Arya, Cheng, and Mount [9], which gives an approximation algorithm for minimum-length tours that use different size cutters, and the paper of Arkin, Held, and Smith [7], which examines the problem of minimizing the number of retractions for “zig-zag” machining without “remilling,” showing that the problem is NP-complete and giving an $O(1)$ -approximation algorithm.

Geometric tour problems with turn costs have been studied by Aggarwal et al. [1], who study the *angular-metric TSP*. The objective is to compute a tour on a set of points, such that the sum of the direction changes at vertices is minimized: For any vertex v_i with incoming edge $v_{i-1}v_i$ and outgoing edge v_iv_{i+1} , the change of direction is given by the absolute value of the angle between $v_{i-1}v_i$ and v_iv_{i+1} . The problem turns out to be NP-hard, and an $O(\log n)$ -approximation is given. Fekete [20] and Fekete and Woeginger [21] have studied a variety of *angle-restricted tour* (ART) problems. Covering problems of a different nature have been studied by Demaine, Demaine, and Mitchell [16], who considered algorithmic issues of origami.

In the operations research literature, there has been an extensive study of arc routing problems, which arise in snow removal, street cleaning, road gritting, trash collection, meter reading, mail delivery, etc.; see the surveys of [10, 18, 19]. Arc routing with turn costs has had considerable attention, as it enables a more accurate modeling of the true routing costs in many situations. Most recently, Clossey, Laporte, and Soriano [13] presented six heuristic methods of attacking arc routing with turn penalties, without resorting to the usual transformation to a TSP problem; however, their results are purely based on experiments and provide no provable performance guarantees. The *directed* postman problem in graphs with turn penalties has been studied by Benavent and Soler [11], who prove the problem to be (strongly) NP-hard and provide heuristics (without performance guarantees) and computational results. (See also Fernández’s thesis [22] and [41] for computational experience with worst-case exponential-time exact methods.)

Our covering tour problem is related to the Chinese postman problem (CPP), which can be solved exactly in polynomial time for “purely” undirected or purely directed graphs. However, the turn-weighted CPP is readily seen to be NP-complete: Hamiltonian cycle in line graphs is NP-complete (contrary to what is reported in [25]; see page 246 of West [45]), implying that the TSP in line graphs is also NP-complete. The CPP on graph G with turn costs at nodes (and zero costs on edges) is equivalent to the TSP on the corresponding line graph, $\mathcal{L}(G)$, where the cost of an edge in $\mathcal{L}(G)$ is given by the corresponding turn cost in G . Thus, the turn-weighted CPP is also NP-complete.

2. Preliminaries. This section formally defines the problems at hand and various special cases of interest.

Problem definitions. The general *geometric milling problem* is to find a closed curve (not necessarily simple) whose Minkowski sum with a given tool (cutter) is precisely a given region P bounded by n edges. In the context of numerically controlled (NC) machines, this region is usually called a *pocket*. Subject to this constraint, we may wish to optimize a variety of objective functions, such as the length of the tour or the number of turns in the tour. We call these problems *minimum-length* and *minimum-turn* milling, respectively. While the latter problem is the main focus of this paper, we are also interested in bicriteria versions of the problem in which both length and number of turns must be small; we also consider the scenario in which the objective function is given by a linear combination of turn cost and distance traveled (see section 5.5).

In addition to choices in the objective function, the problem version depends on the constraints on the tour. The most general case arises when considering a tour that has to visit a discrete set of vertices, connected by a set of edges, with a specified turn cost at each vertex to change from one edge to the next. More precisely, at each vertex, the tour has the choice of (0) going “straight,” if there is one “collinear” edge with

the one currently used (costing no turn), (1) turning onto another, noncollinear edge (costing one turn), or (2) “U-turning” back onto the source edge (costing two turns). Which pairs of the d edges incident to a degree- d vertex are considered “collinear” are specified by a matching in the complete graph K_d : three incident edges cannot be “collinear.” We call this graph-theoretic abstraction the *discrete milling problem*, indicating the close relationship to other graph-theoretic tour optimization problems. We are able to give a number of approximation algorithms for discrete milling that depend on some graph parameters: δ denotes the maximum degree of a vertex and ρ denotes the maximum number of distinct “directions” coming together at a vertex. (For example, for graphs arising from d -dimensional grids, these values are bounded by $2d$ and d , respectively.)

A special case of discrete milling arises when dealing with “thin” structures in two- or three-dimensional space, where the task is to travel all of a given set of “channels,” which are connected at vertices. This resembles a CPP, in that it requires us to travel a given set of edges; however, in addition to the edge cost, there is a cost at the vertices when moving from one edge to the next. For this scenario, we are able to describe approximation factors that are independent of other graph parameters.

More geometric problems arise when considering the milling of a polygonal region P . In the *orthogonal milling problem*, the region P is an orthogonal polygonal domain (with holes) and the tool is an (axis-parallel) unit-square cutter constrained to axis-parallel motion, with edges of the tour alternating between horizontal and vertical. All turns are orthogonal; 90° turns incur a cost of 1, while a “U-turn” has a cost of 2. In the *integral orthogonal* case, all coordinates of boundary edges are integers, so the region can be considered to be the (connected) union of N *pixels*, i.e., axis-parallel unit squares with integer vertices. Note that in general, N may not be bounded by a polynomial in n . Instead of dealing directly with a geometric milling problem, we often find it helpful to consider a more combinatorial problem, and then adapt the solution back to the geometric problem. In particular, for integral orthogonal milling, we may assume that an optimal tour can be assumed to have its vertex coordinates of the form $k + \frac{1}{2}$ for integral k . Then, milling in an integral orthogonal polygon (with holes) is equivalent to finding a tour of all the vertices (“pixels”) of a grid graph; see Figure 2.1.

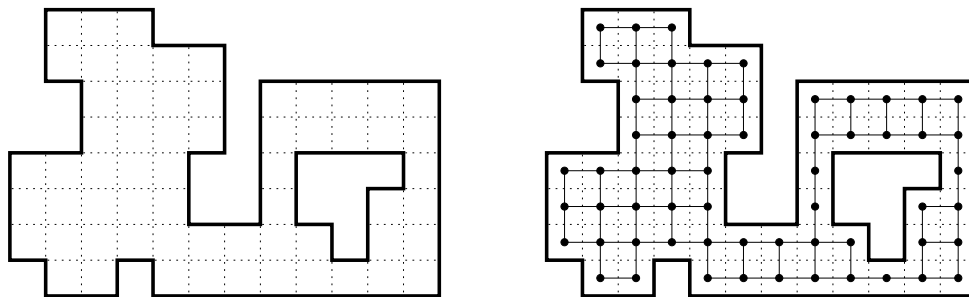


FIG. 2.1. An instance of the integral orthogonal milling problem (left) and the grid graph model (right).

An interesting special case of integral orthogonal milling is the *thin orthogonal milling problem*, in which the region does not contain a 2×2 square of pixels. This is also closely related to discrete milling, as we can think of edges embedded into the planar grid, such that vertices and channels are well separated. This problem of

finding a tour with minimum turn cost for this class of graphs is still NP-complete, even for a subclass for which the corresponding problem of minimizing total distance is trivial; this highlights the particular difficulty of dealing with turn cost. On the other hand, thin orthogonal milling allows for particularly fast and efficient approximation algorithms.

Other issues. It should be stressed that using turn cost instead of (or in addition to) edge length changes several characteristics of distances. One fundamental problem is illustrated by the example in Figure 2.2: the triangle inequality does not have to hold when using turn cost. This implies that many classical algorithmic approaches for graphs with nonnegative edge weights (such as using optimal 2-factors or the Christofides method for the TSP) cannot be applied without developing additional tools.

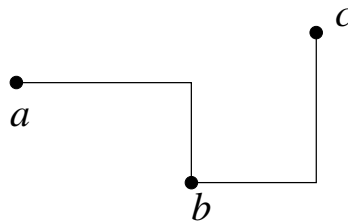


FIG. 2.2. *The triangle inequality may not hold when using turn cost as distance measure: $d(a, c) = 3 > 2 = d(a, b) + d(b, c)$.*

In fact, in the presence of turn costs we distinguish between the terms *2-factor*, i.e., a set of edges, such that every vertex is incident to two of them, and *cycle cover*, i.e., a set of cycles, such that every vertex is covered. While the terms are interchangeable when referring to the set of edges that they constitute, we make a distinction between their respective *costs*: a “2-factor” has a cost consisting of the sum of edge costs but does not necessarily account for the turn cost between its two incident edges, while the cost of a “cycle cover” includes also the turn costs at vertices.

It is often useful in designing approximation algorithms for optimal *tours* to begin with the problem of computing an optimal *cycle cover*, minimizing the total number of turns in a set of cycles that covers P . Specifically, we can decompose the problem of finding an optimal (minimum-turn) tour into two tasks: finding an optimal cycle cover, and merging the components. Of course, these two processes may influence each other: there may be several optimal cycle covers, some of which are easier to merge than others. (In particular, we say that a cycle cover is *connected* if the graph induced by the set of cycles and their intersections is connected.) As we will show, even the problem of optimally merging a connected cycle cover is NP-complete. This is in contrast to minimum-length milling, where an optimal connected cycle cover can trivially be converted into an optimal tour that has the same cost.

Algorithms whose running time is polynomial in the explicit encoding size (pixel count) are *pseudopolynomial*. Algorithms whose running time is polynomial in the implicit encoding size are *polynomial*. This distinction becomes an important issue when considering different ways to encode input and output; e.g., a large set of pixels forming an $a \times b$ rectangle can be described in space $O(\log a + \log b)$ by simply describing the bounding edges, instead of listing all ab individual pixels. In integral orthogonal milling, one might think that it is most natural to encode the grid graph with vertices, because the tour will be embedded on this graph and will, in general,

have complexity proportional to the number N of pixels. But the input to any geometric milling problem has a natural encoding by specifying only the n vertices of the polygon P . In particular, long edges are encoded in binary (or with one real number, depending on the model) instead of unary. It is possible to get a running time depending only on this size, but of course we need to allow for the output to be encoded *implicitly*. That is, we cannot explicitly encode each vertex of the tour because there are too many (the number can be arbitrarily large even for a succinctly encodable rectangle). Instead, we encode an abstract description of the tour that is easily decoded.

Finally, we mention that many of our results carry over from the *tour* (or cycle) version to the *path* version, in which the cutter need not return to its original position. In this paper, we omit the straightforward changes necessary to compute optimal paths. A similar adjustment can be made for the related case of *lawn mowing*, in which the sweep of the cutter is allowed to go outside P during its motion. Clearly, our techniques are also useful for scenarios of this type.

3. NP-completeness. Arkin, Fekete, and Mitchell [6] have proved that the problem of optimizing the length of a milling tour is NP-hard. Their proof is based on the well-known hardness of deciding whether a grid graph has a Hamiltonian cycle [29, 31]. This result implies that it is NP-hard to find a tour of minimum total length that visits all vertices. If, on the other hand, we are given a connected cycle cover of a graph that has minimum total length, then it is trivial to convert it into a tour of the same length by merging the cycles into one tour.

In this section we show that if the quality of a tour is measured by counting *turns*, then even this last step of turning an optimal connected cycle cover into an optimal tour is NP-complete. Thus we prove that it is NP-hard to find a milling tour that optimizes the number of turns for a polygon with holes.

THEOREM 3.1. *Minimum-turn milling is NP-complete, even when we are restricted to the orthogonal thin case, and are already provided with an optimal connected cycle cover.*

Because thin orthogonal milling is a special case of thin milling as well as orthogonal milling, and because it is easy to convert an instance of thin orthogonal milling into an instance of integral orthogonal milling, we have the following.

COROLLARY 3.2. *Discrete milling, orthogonal milling, and integral orthogonal milling are NP-complete.*

Proof of Theorem 3.1. Our reduction proceeds in two steps. First we show that the problem *Hamiltonicity of unit segment intersection graphs* (HUSIG) of deciding the Hamiltonicity of intersection graphs of axis-parallel unit segments is hard. To see this, we use the NP-hardness of deciding Hamiltonicity of grid graphs [29, 31] and argue that any grid graph can be represented in this form (see Figure 3.1).

Consider a set of integer grid points that induce a grid graph G . Note that G is bipartite, because one can 2-color the nodes by coloring a grid point (x, y) black (resp., white) if $x + y$ is odd (resp., even). After rotating the point set by $\pi/4$, the coordinate of each point is an integer multiple of $1/\sqrt{2}$. Scaling down the resulting arrangement by a factor of $3/\sqrt{2}$ results in an arrangement in which the coordinate of each point is an integer multiple of $1/3$, and the shortest distance between two points of the same color class is $2/3$. For the resulting set of points $p_i = (x_i, y_i)$, let $p'_i = p_i + (\varepsilon_i, \varepsilon_i)$ be given as the set obtained by “perturbations” ε_i that are small and all distinct. Then represent each “white” vertex by a horizontal unit segment centered at p'_i and each “black” vertex by a vertical unit segment centered at p'_i . Now

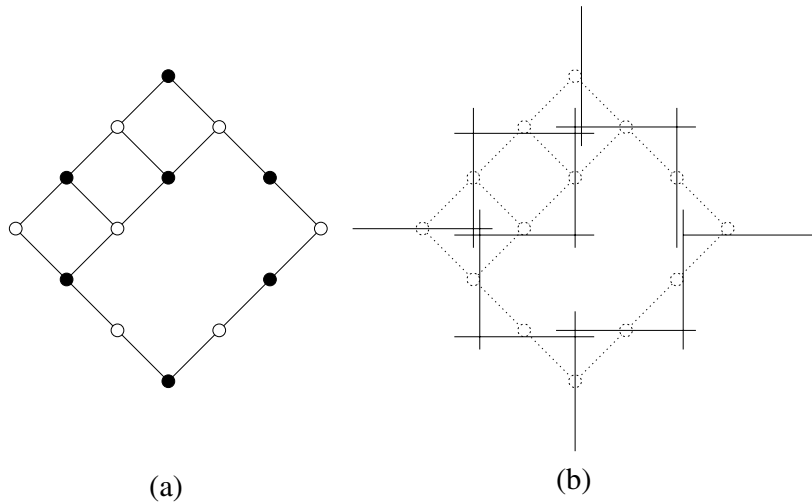


FIG. 3.1. (a) A grid graph G . (b) A representation of G as an intersection graph of axis-parallel unit segments.

it is easy to see that the resulting unit segment intersection graph is precisely the original grid graph G .

In a second step, we show that the problem HUSIG reduces to the problem of milling with turn costs. The outline of our argument is illustrated in Figure 3.2.

Consider a unit segment intersection graph G , given by a set of axis-parallel unit segments, as shown in Figure 3.2(a). Figure 3.2(b) shows the corresponding graph, with a Hamiltonian cycle indicated in bold. Without loss of generality, we may assume that G is connected. Let s be the number of nodes of G .

As shown in Figure 3.2(c), we replace each line segment by a cycle of four thin axis-parallel corridors. This results in a connected polygonal region P having $4s$ convex corners. Clearly, any cycle cover or tour cover of P must have at least $4s$ turns; by using a cycle for each set of four corridors representing a strip, we get a cycle cover \mathcal{C} with $4s$ turns. Therefore, \mathcal{C} is an optimal cycle cover, and it is connected, because G is connected.

Now assume that G has a Hamiltonian cycle. It is easy to see (Figure 3.2(f)) that this cycle can be used to construct a milling tour of P with a total of $5s$ turns: Each time the Hamiltonian cycle moves from one vertex v_i of the grid graph to the next vertex v_j , the milling tour moves from the cycle C_i representing v_i to the cycle C_j representing v_j , at an additional cost of 1 turn for each of the s edges in the Hamiltonian cycle.

Assume conversely that there is a milling tour T with at most $5s$ turns. We refer to turns at the corners of 4-cycles as *convex turns*. The other turns are called *crossing turns*.

As noted above, the convex corners of P require at least $4s$ convex turns. Consider the sequence of turns t_1, \dots, t_{5s} in T . By construction, the longest contiguous subsequence of convex turns contains at most four different convex turns. (More precisely, we can have such a subsequence with four different convex corners only if these four corners belong to the same 4-cycle representing a unit segment.) Furthermore, we need at least one additional crossing turn at an interior crossing of two corridors to get from one convex corner to another convex corner not on the same 4-cycle. (More

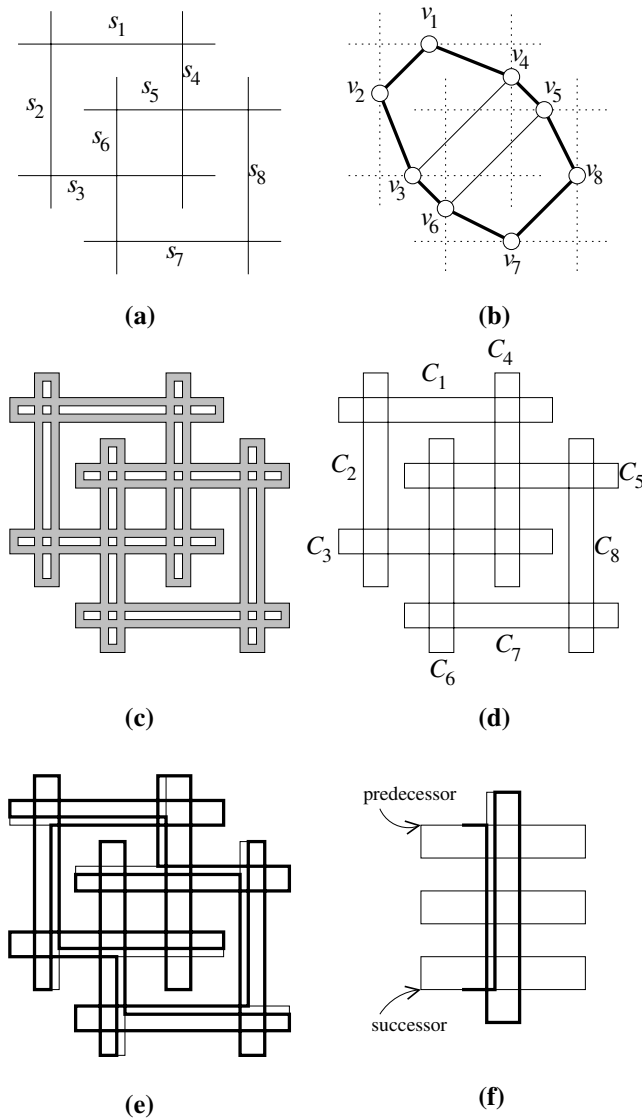


FIG. 3.2. Thin orthogonal milling with turn cost is NP-hard: (a) a set of $s = 8$ axis-parallel unit segments, denoted by s_1, \dots, s_8 ; (b) the corresponding intersection graph G , with the Hamiltonian cycle $v_1, v_2, v_3, v_6, v_7, v_8, v_5, v_4$ shown in bold; (c) representing G by a connected region consisting of $4s$ corridors; (d) a drawing of the graph induced by the instance of thin orthogonal milling, with the $s = 8$ rectangular cycles C_1, \dots, C_8 ; (e) a milling tour with $5s$ turns corresponding to the Hamiltonian cycle in G ; (f) milling the four corridors of a cycle using five turns.

precisely, one crossing turn is sufficient only if the two connected convex corners belong to 4-cycles representing intersecting unit segments.) Therefore, we need at least c crossing turns if we have at least c contiguous subsequences as described above. This means that $c \geq s$; hence, $c = s$ by the assumption on the number of turns on T . Because the c crossing turns correspond to a closed round trip in G that visits all s vertices, this implies that we have a Hamiltonian cycle, concluding the proof. \square

4. Approximation tools. There are three main tools that we use to develop approximation algorithms: computing optimal cycle covers for milling the “boundary” of P (section 4.1), converting cycle covers into tours (section 4.2), and using optimal (or nearly optimal) “strip covers” (section 4.3). In this section, our description mostly focuses on orthogonal milling; however, we will see in the following section 5.1 how some of our tools can also be applied to the general case of discrete milling.

4.1. Boundary cycle covers. Consider first the problem of finding a minimum-turn cycle cover for covering a certain subset, \bar{P} , of P that is along its boundary. This will turn out to be a useful tool for approximation algorithms. Specifically, in orthogonal milling we define the set \bar{P} of *boundary pixels* to consist of pixels that have at least one of their four edges on a boundary edge of the polygon; i.e., in the grid graph that describes adjacency of pixels, these are pixels of degree at most 3. Let $N_{\bar{P}}$ be the number of boundary pixels. A *boundary cycle cover* is a collection of cycles that visit all boundary pixels.

We define an auxiliary structure, $G_{\bar{P}} = (V_{\bar{P}}, E_{\bar{P}})$, which is a complete weighted graph on $2N_{\bar{P}}$ vertices; for ease of description, we will refer to $G_{\bar{P}}$ as a set of points and paths between them. This will allow us to map boundary cycle covers in P to matchings of corresponding turn cost in $G_{\bar{P}}$. For this purpose, map each pixel $p_i \in \bar{P}$ to two vertices in $V_{\bar{P}}$, $v_i^{(0)}$ and $v_i^{(1)}$. For each boundary pixel p_i , this pair represents an orientation that is attained by a cutter when visiting p_i . Depending on the boundary structure of p_i , there are four different cases; refer to Figure 4.1.

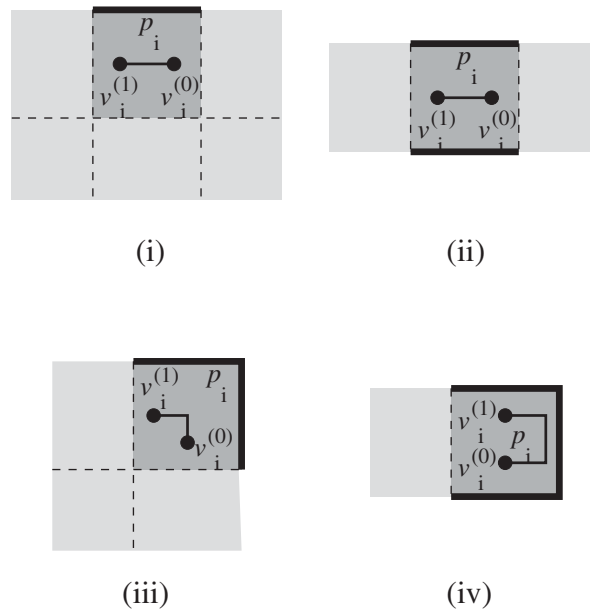


FIG. 4.1. Representing a boundary pixel p_i by a pair of vertices $v_i^{(0)}$ and $v_i^{(1)}$.

- (i) One edge of p_i is a boundary edge of the polygon.
- (ii) Two opposite edges of p_i are boundary edges of the polygon.
- (iii) Two adjacent edges of p_i are boundary edges of the polygon.
- (iv) Three edges of p_i are boundary edges of the polygon.

For easier description, we refer to the vertices $v_i^{(0)}$ and $v_i^{(1)}$ as points embedded

within p_i , as shown in Figure 4.1. Furthermore, we add a *mandatory path* m_i between $v_i^{(0)}$ and $v_i^{(1)}$, represented by a polygonal path with $c(m_i) = 0$ (cases (i) and (ii)), $c(m_i) = 1$ (case (iii)), or $c(m_i) = 2$ turns (case (iv)), as shown in the figure. This path maps the contour of P at p_i , and it represents orientations that a cutter has to attain when visiting pixel p_i . Note that traveling from $v_i^{(h)}$ to $v_j^{(1-h)}$ along m_i induces a heading $\delta_i^{(h,-)}$ when leaving $v_i^{(h)}$ and a heading $\delta_i^{(1-h,+)}$ when arriving at $v_i^{(1-h)}$. Note that $\delta_i^{(h,-)}$ is opposite to $\delta_i^{(h,+)}$.

Now we add a set of *optional paths*, representing the weighted edges $E_{\overline{P}}$ of the complete graph $G_{\overline{P}}$. For an example, refer to Figure 4.2. For any pair of vertices $v_i^{(h)}$ and $v_j^{(k)}$, let $d(v_i^{(h)}, v_j^{(k)})$ be the minimum number of turns necessary when traveling from $v_i^{(h)}$ with heading $\delta_i^{(h,+)}$ to $v_j^{(k)}$ with heading $\delta_j^{(k,-)}$. Note that $d(v_i^{(h)}, v_j^{(k)}) = d(v_j^{(k)}, v_i^{(h)})$, as any shortest path can be traveled in the opposite direction. Using a Dijkstra-like approach, we can compute these distances from one boundary pixel to all other boundary pixels in time $O(N_{\overline{P}} \log N_{\overline{P}})$; see the overview in [39] or the paper [37]. The overall time of $O(N_{\overline{P}}^2 \log N_{\overline{P}})$ for computing all these link distances is dominated by the following step: In time $O(N_{\overline{P}}^3)$ [24, 43], find a minimum-weight perfect matching in the complete weighted graph $G_{\overline{P}}$.

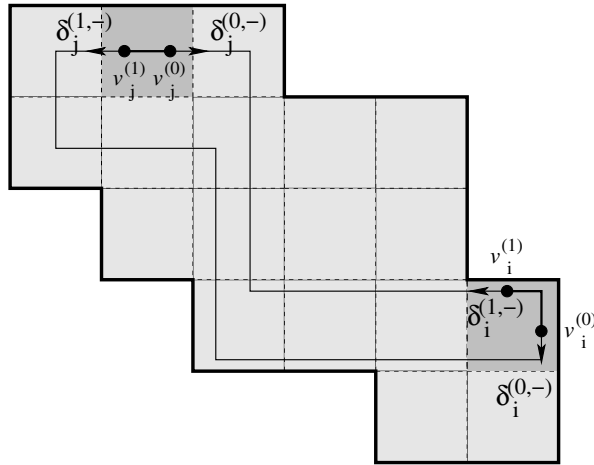


FIG. 4.2. The cost of traveling between two pixels: $d(v_i^{(1)}, v_j^{(0)}) = d(v_j^{(0)}, v_i^{(1)}) = 2$, while $d(v_i^{(0)}, v_j^{(1)}) = d(v_j^{(1)}, v_i^{(0)}) = 5$.

Now it is not hard to see the following.

LEMMA 4.1. Any boundary cycle cover in P with t turns can be mapped to a perfect matching in $G_{\overline{P}}$ of cost $t - \sum_{p_i \in \overline{P}} c(m_i)$, and vice versa.

Proof. Whenever a boundary cycle cover visits a boundary pixel p_i , it has to perform the turns corresponding to the mandatory path m_i . Moreover, moving from one pixel p_i to the next pixel p_j can be mapped to the optional path corresponding to the edges $(v_i^{(h)}, v_j^{(k)})$; clearly, the overall cost is as stated.

Conversely, it is straightforward to see that the combination of a perfect matching in $G_{\overline{P}}$ and the mandatory paths yields a boundary cycle cover of corresponding cost. \square

Using the algorithms described above, we also obtain the following.

THEOREM 4.2. *Given the set of $N_{\overline{P}}$ boundary pixels, a minimum-turn boundary cycle cover can be computed in time $O(N_{\overline{P}}^3)$, the time it takes to compute a perfect matching in $G_{\overline{P}}$.*

If the set of pixels is not given in unary, but implicitly as the pixels contained in a region with n edges, the above complexity is insufficient. However, we can use local modifications to argue the following tool for speeding up the search for an optimal perfect matching.

LEMMA 4.3. *Let p_i and p_j be neighboring boundary pixels that are adjacent to the same boundary edge, so $d(v_i^{(h)}, v_j^{(k)}) = 0$ for an appropriate choice of h and k . Then there is an optimal matching containing $(v_i^{(h)}, v_j^{(k)})$.*

Proof. This follows by a simple exchange argument. See Figure 4.3. Suppose two adjacent pixels p_i and p_j along the same boundary edge are not matched to each other, let $v_i^{(h)}$ be the vertex such that $\delta_i(h, -)$ is heading for p_j , and let $v_j^{(k)}$ be the vertex such that $\delta_j(k, -)$ is heading for p_i . Furthermore suppose that $v_{i'}^{(h')}$ is matched to $v_i^{(h)}$ and $v_{j'}^{(k')}$ is matched to $v_j^{(k)}$. Then we can match $v_{i'}^{(h')}$ to $v_{j'}^{(k')}$ and $v_i^{(h)}$ to $v_j^{(k)}$ without changing the cost of the matching. \square

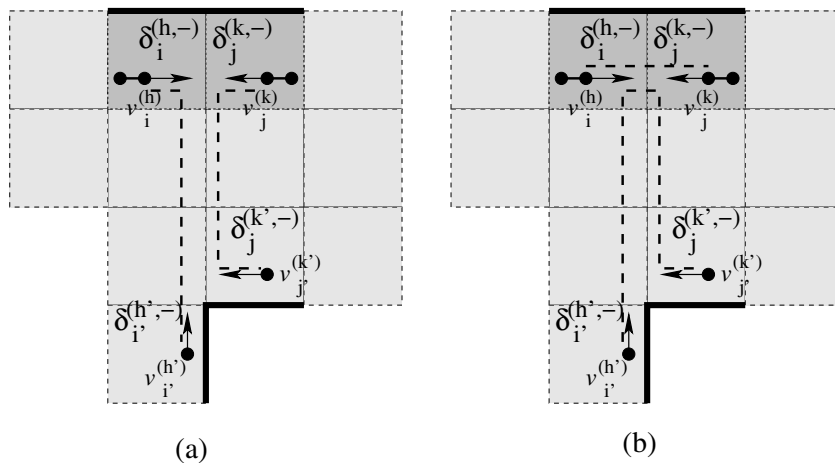


FIG. 4.3. By performing local modifications, an optimal cycle cover can be assumed to cover each collinear piece of the boundary in one connected strip.

This allows us to obtain a strongly polynomial version of the matching algorithm of Theorem 4.2.

THEOREM 4.4. *A minimum-turn boundary cycle cover can be computed in time $O(n^3)$.*

Proof. By applying Lemma 4.3 repeatedly, we get $O(n)$ connected boundary strips, consisting of sets of collinear boundary pixels. These can be determined efficiently by computing offsets of the boundary edges. This leaves only $O(n)$ endpoints of such strips to be matched, resulting in the claimed complexity. \square

Note that the validity of this argument is not restricted to the *integral* orthogonal case, but remains valid even for orthogonal regions with arbitrary boundary edges.

Remark. The definition of the “boundary” pixels \overline{P} used here does not include all pixels that touch the boundary of P in a diagonal fashion; in particular, it omits the “reflex pixels” that share a corner, but no edge, with the boundary of P . It seems

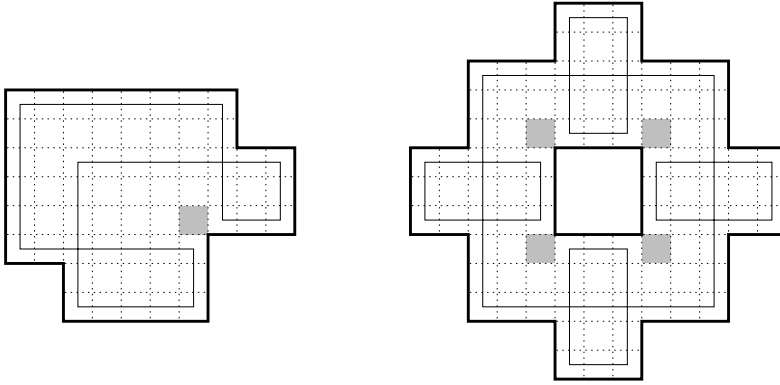


FIG. 4.4. *Optimally covering all pixels that have an edge against the boundary can leave reflex pixels uncovered.*

difficult to require that the cycle cover mill reflex pixels, because Lemma 4.3 does not extend to this case, and an optimal cycle cover of the boundary (as defined above) may have fewer turns than an optimal cycle cover that mills the boundary \bar{P} plus the reflex pixels; see Figure 4.4.

4.2. Merging cycles. It is often easier to find a minimum-turn cycle cover (or constant-factor approximation thereof) than to find a minimum-turn tour. We show that an exact or approximate minimum-turn cycle cover implies an approximation for a minimum-turn tour.

We concentrate on the integral orthogonal case. First we define a few terms precisely. Two pixels are *adjacent* if the distance between their centers is 1. Two cycles T_1, T_2 are *intersecting* if and only if $T_1 \cap T_2 \neq \emptyset$. Two cycles are called *touching* if and only if they are not intersecting and there exist pixels $p_1 \in T_1, p_2 \in T_2$ such that p_1 and p_2 are adjacent.

LEMMA 4.5. *Let P_1 and P_2 be two cycles, with t_1 and t_2 turns, respectively, and let p be a pixel that is contained in both cycles. Then there is a cycle milling the union of pixels milled by P_1 and P_2 and having at most $t_1 + t_2 + 2$ turns. This cycle can be found in time linear in the number of its turns.*

Proof. Let the neighbors of p in P_1 be a_1, a_2 and those of p in P_2 be b_1, b_2 . Connect a_1 via p to b_1 and a_2 via p to b_2 to get the required tour. The two connections may add at most a turn each. Hence the resulting tour can be of size at most $t_1 + t_2 + 2$. \square

LEMMA 4.6. *Given two touching cycles T_1, T_2 with t_1, t_2 turns, respectively, there is a tour T with at most $t_1 + t_2 + 2$ turns that mills the union of pixels milled by T_1, T_2 .*

Proof. Because T_1, T_2 are touching, $T_1 \cap T_2 = \emptyset$ and there exist adjacent pixels $p_1 \in T_1$ and $p_2 \in T_2$. Without loss of generality assume that p_2 is a leftmost such pixel, and below p_1 . Due to these constraints, T_2 can enter/exit p_2 from only two sides. Hence there are only three ways in which T_2 can visit p_2 . These are shown in Figure 4.5. For all three ways we show in Figure 4.5 how to cut and extend tour T_2 without adding any extra turns, to get a path P_2 starting and ending at pixel p_1 . Cut T_1 at p_1 to get a path P_1 . By possibly adding two turns, we can merge the two paths into one tour. \square

With the help of these lemmas, we deduce the following.

THEOREM 4.7. *A cycle cover with t turns can be converted into a tour with at*

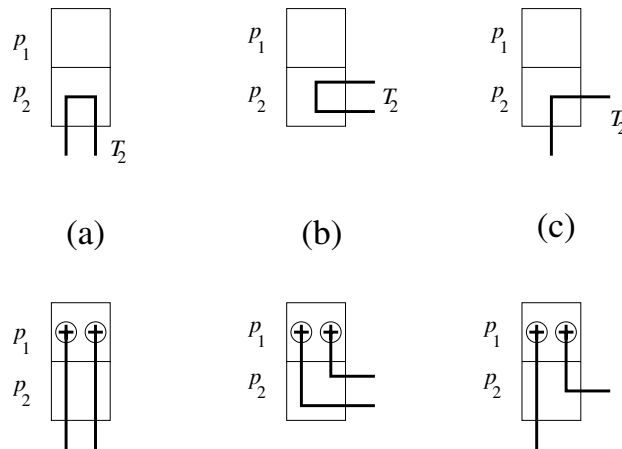


FIG. 4.5. *Merging two touching tours: There are three possible ways of tour T_2 visiting pixel p_2 (above). In each case, we can modify T_2 into a path that visits pixel p_1 (below); at a cost of possibly one extra turn at each end of the path, we can merge it with tour T_1 .*

most $t + 2(c - 1)$ turns, where c is the number of cycles.

Proof. We prove this theorem by induction on the number of tours, c , in the cycle cover. The theorem is trivially true for $c = 1$. For any other c , choose any $c - 1$ cycles with a total of t' turns and find a tour T' that covers those $c - 1$ cycles; by induction, it has $t' + 2(c - 1)$ turns. Let the remaining cycle, R , have r turns. Thus $t = t' + r$. Because the polygon is connected, the set of pixels milled by R and T' must be connected. Hence either T' and R are intersecting or touching. By Lemmas 4.5 and 4.6 we can merge R and T' into a single tour T with at most $t' + 2(c - 1) + r + 2$ turns, i.e., $t + 2c$ turns. \square

COROLLARY 4.8. *A cycle cover of a connected rectilinear polygon with t turns can be converted into a single milling tour with at most $\frac{3}{2}t$ turns.*

Proof. This follows immediately from Theorem 4.7 and the fact that each cycle has at least four turns. \square

Unfortunately, general merging is difficult (as illustrated by the NP-hardness proof of Theorem 3.1), so we cannot hope to improve these general merging results by more than a constant factor.

4.3. Strip and star covers. A key tool for approximation algorithms is a covering of the region by a collection of “strips.” A *strip* is a maximal straight segment whose Minkowski sum with the tool is contained in the region. A *strip cover* is a collection of strips whose Minkowski sums with the tool cover the entire region. A *minimum strip cover* is a strip cover with the fewest strips.

LEMMA 4.9. *The size of a minimum strip cover is a lower bound on the number of turns in a cycle cover (or tour) of the region.*

Proof. Any cycle cover induces a strip cover by extending each edge to have maximal length. The number of strips in this cover equals the number of turns in the cycle cover. \square

In the discrete milling problem, a related notion is a “rook placement.” A *rook* is a marker placed on a pixel, which can *attack* every pixel to which it is connected via a straight axis-parallel path inside the region. A *rook placement* is a collection of rooks, no two of which can attack each other. See Figure 4.6 for an illustration; this

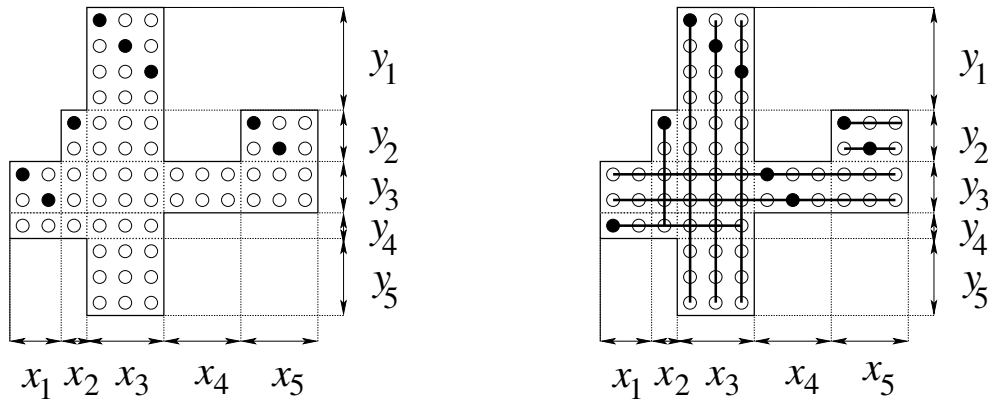


FIG. 4.6. (Left) An orthogonal region, its subdivision into axis-parallel strips, and a resulting greedy rook cover (indicated by black pixels). (Right) An optimal strip cover, and an optimal rook cover (indicated by black pixels).

tool will be used in Theorem 5.6, based on the following lemma.

LEMMA 4.10. *The size of a maximum rook placement is a lower bound on the number of turns in a cycle cover (or tour) for discrete milling.*

Proof. Consider a rook placement and a cycle cover of the region, which must in particular cover every rook. Suppose that one of the cycles visits rooks q_1, \dots, q_k in that order. No two rooks can be connected by a single straight axis-parallel line segment, so the cycle must turn between each rook, for a total of at least k turns. Because each rook is traversed by at least one cycle, the number of turns (and hence the number of segments in a tour) is at least the number of rooks. \square

In the integral orthogonal milling problem, the notions of strip cover and rook placement are dual and efficient to compute.

LEMMA 4.11. *For integral orthogonal milling, a minimum strip cover and a maximum rook placement have equal size. For a polygonal region with n edges and N pixels they can be computed in time $O(N^{2.376})$ or $O(n^{2.5} \log N)$.*

Proof. For the case of $N \in O(n)$, the claim follows from Proposition 2.2 in [26]: We rephrase the rook-placement problem as a matching problem in a bipartite graph $G = (V_1, V_2, E)$. Let the vertices in V_1 correspond to vertical strips, and let the vertices in V_2 correspond to horizontal strips. An edge $e = (u, v) \in E$ exists if the vertical strip corresponding to u and the horizontal strip corresponding to v have a pixel in common (i.e., the strips cross). It is easy to see that a maximum-cardinality matching in this bipartite graph corresponds to a rook placement: each edge (u, v) in the matching corresponds to the unique pixel that vertical strip u and horizontal strip v have in common.

Similarly, observe that the minimum strip-cover problem is equivalent to a minimum vertex-cover problem in the bipartite graph defined above. Each strip in the strip cover defines a vertex in the vertex cover. The requirement that each pixel must be covered by at least one strip is equivalent to the requirement that each edge of the graph must be covered by at least one vertex.

By the famous König–Egerváry theorem, the maximum cardinality matching in a bipartite graph is equal in size to the minimum vertex cover, and therefore both can be solved in time polynomial in the size of the graph; more precisely, this can be achieved in time $O(N^\omega)$, the time needed for multiplying two $N \times N$ matrices, for

example, $\omega = 2.376$; see the paper [28], or the survey in chapter 16 of [43], which also lists other, more elementary methods.

To get the claimed running time even for “large” N , using implicit encoding, we decompose the region into “thick strips” by conceptually coalescing adjacent horizontal strips with the same horizontal extent, and similarly for vertical strips. In other words, thick strips are bounded by two vertices of the region, and hence there are only $O(n)$ of them. We define the same bipartite graph but add a weight to each vertex corresponding to the width of the strip (i.e., the number of strips coalesced). Instead of a matching, in which each edge of the graph is either included in the matching or not, we now have a multiplicity for each edge, which is the minimum of the weights of its two endpoints. The interpretation is that an edge corresponds to a rectangle in the region (the intersection of two thick strips), and the number of rooks that can be placed in such a rectangle is at most the minimum of its width and height.

The weighted-matching problem we consider is that each edge can be included in the matching with a multiplicity up to its weight. Furthermore, the sum of the included multiplicities of edges incident to a vertex cannot exceed the weight of the vertex. A weighted version of the König–Egerváry theorem states that the minimum-weight vertex cover is equal to the maximum-weight matching. (This weighted version can be easily proved using the max-flow min-cut theorem.) Both problems can be solved in polynomial time using a max-flow algorithm, on a modified graph in which a source vertex s is added with edges to all vertices in V_1 , of capacity equal to the weight of the vertex, and a sink vertex t is added with edges to it from all vertices in V_2 with capacity equal to the vertex capacity. Edges between V_1 and V_2 are directed from V_1 and have capacity equal to the weight of the edge. Currently, the best known running time is $O(\sqrt{nm} \log nW)$ for a bipartite graph with n vertices, m edges, and maximum weight W [23, 43]. For our purposes, this yields a complexity of $O(n^{2.5} \log N)$. \square

Note that using weights on the edges is crucial for the correctness of our objective; moreover, this has a marked effect on the complexity of the problem: Finding a minimum number of axis-parallel rectangles (regardless of their size) that covers an integral orthogonal polygon is known to be an NP-complete problem, even for the case of polygon without holes [15].

For general discrete milling, it is possible to approximate an optimal strip cover as follows. Greedily place rooks until no more can be placed (i.e., until there is no unattackable vertex). This means that every vertex is attackable by some rook, so by replacing each rook with all possible strips through that vertex, we obtain a strip cover of size ρ times the number of rooks, where ρ is the maximum degree of the underlying graph. (We call this type of strip cover a *star cover*.) But each strip in a minimum strip cover can only cover a single rook, so this is a ρ -approximation to the minimum strip cover. We have thus proved the following.

LEMMA 4.12. *In discrete milling, the number of stars in a greedy star cover is a lower bound on the number of strips, and hence serves as a ρ -approximation algorithm for minimum strip covers. Computing a greedy star cover can be done in time $O(N)$.*

Proof. Loop over the vertices of the underlying graph. Whenever an unmarked vertex is found, add it to the list of rooks, and mark it and all vertices attackable by it. Now convert each rook into a star as in the proof of Lemma 4.10. Each edge is traversed only once during this process. \square

5. Approximation algorithms. We employ four main approaches to building approximation algorithms, repeatedly in several settings:

(i) *Star cover + doubling + merging.*

The simplest but most generally applicable idea is to cover the region by a collection of stars. “Doubling” these stars results in a collection of cycles, which can then be merged into a tour using general techniques.

(ii) *Strip cover + doubling + merging.*

Tighter bounds can be achieved by covering directly with strips instead of stars. Similar doubling and merging steps follow.

(iii) *Strip cover + perfect matching of endpoints + merging.*

The covering of the region is done by the strip cover. To connect these strips into cycles, we find a minimum-weight perfect matching on their endpoints. This results in a cycle cover, which can be merged into a tour using general techniques.

(iv) *Boundary tour + strip cover + perfect matching of odd-degree vertices.*

Again coverage is by a strip cover, but the connection is done differently. We add a tour of the boundary (by merging an optimal boundary cycle cover) and attach each strip to this tour on each end. The resulting graph has several degree-3 vertices, which we fix by adding a minimum-weight matching on these vertices.

5.1. Discrete milling. As described in the preliminaries, we consider two scenarios: While general discrete milling focuses on vertices (and thus resembles the TSP), thin discrete milling requires traveling a set of edges, making it similar to the CPP.

5.1.1. General discrete milling. Our most general approximation algorithm for the discrete milling problem runs in linear time. First we take a star cover according to Lemma 4.12, which approximates an optimal strip cover to within a factor of ρ . Then we tour the stars using an efficient method described below. Finally we merge these tours using Theorem 4.7.

We tour each star emanating from a vertex v using the following method—see Figure 5.1. Consider a strip s in the star, and suppose its ends are the vertices u_i and u_j . A strip having both of its endpoints distinct from v is called a *full strip*; a strip one of whose endpoints is equal to v is called a *half strip*. Half strips are covered by three edges, (v, u_j) , (u_j, u_j) , and (u_j, v) , making a U-turn at endpoint u_j . (This covering is shown for the half strip (v, u_1) in Figure 5.1(b).) Full strips are covered by five edges, (v, u_i) , (u_i, u_i) , (u_i, u_j) , (u_j, u_j) , and (u_j, v) , with U-turns at both ends, u_i and u_j . (This covering is shown for the full strip (u_5, u_2) in Figure 5.1(b).) Now we have several paths of edges starting and ending at v . By joining their ends we can easily merge these paths into a cycle.

The number of turns in this cycle is 3 times the number of half strips, plus 5 times the number of full strips. This is equivalent to the number of distinct directions at v plus 2 times the degree of v . (The number of directions at v is equal to the number of full strips plus the number of half strips, by definition. The degree of v is equal to 2 times the number of full strips plus the number of half strips.) Lemma 4.12 implies that the number of stars is a lower bound on the number of turns in a cycle cover of the region, proving the following.

THEOREM 5.1. *There is an $O(N)$ -time $(2\delta + \rho)$ -approximation for finding a minimum-turn cycle cover in discrete milling. Furthermore, the maximum coverage of a vertex (i.e., the maximum number of times a vertex is swept) is δ , and the cycle cover is a δ -approximation on length.*

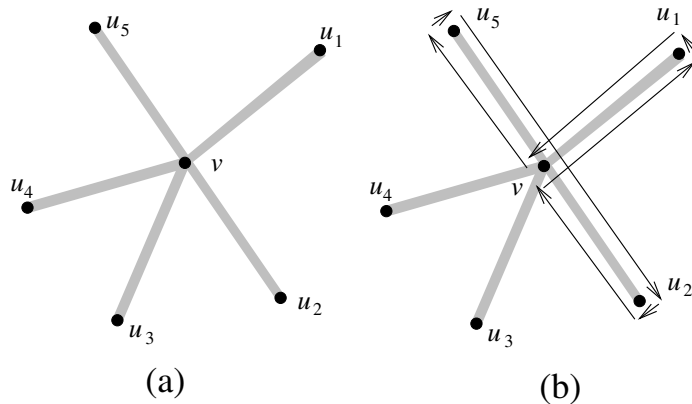


FIG. 5.1. (a) A star of degree 5 around vertex v . $(v, u_1), (v, u_3), (v, u_4)$ are half strips, and (u_2, u_5) is a full strip. (b) A covering with three edges for a half strip, and a covering with five edges for the full strip.

Proof. As the star cover, by definition, contains all vertices, the cycle cover obtained by traversing the stars also does. As stated above, the number of turns in each cycle covering a star is the number of directions at v , plus 2 times the degree of v . Summing over all stars, we get the claimed approximation bound. The running time follows directly from Lemma 4.12. Deriving the values for maximum coverage and overall length is straightforward. \square

COROLLARY 5.2. *There is a linear-time $(2\delta + \rho + 2)$ -approximation for minimum-turn discrete milling. Furthermore, the maximum coverage of a vertex is δ , and the tour is a δ -approximation on length.*

Proof. We apply Theorem 4.7. The number of cycles to be merged is the number of stars, which by Lemma 4.12 is a lower bound on the number of turns in a tour of the region. We pay at most two turns per cycle for the merge. There is no additional cost of length due to the merge, as the stars form a connected graph. \square

5.1.2. Thin discrete milling. As described in section 2, a more special structure arises if the structure to be milled consists of a connected set of “channels” that have to be milled. In this case, achieving a strip cover is trivial.

LEMMA 5.3. *In thin discrete milling a strip cover can be obtained in linear time by merging edges that are collinear at some vertex.*

Using method (ii) described at the beginning of the section, we get the following approximation results.

THEOREM 5.4. *There is a 4-approximation of complexity $O(n \log n)$ for computing a minimum-turn cycle cover for a graph with n edges, and a 6-approximation of the same complexity for computing minimum-turn tours.*

Proof. Clearly, the number of strips is a lower bound on the cost of any cycle cover or tour. Turning each strip into a cycle with two U-turns, i.e., 4 turns, yields a cycle cover within a factor 4 of the optimum. Merging these cycles at a cost of 2 turns per merge yields a tour within a factor of 6 times the optimum, as each cycle has 4 turns. \square

Clearly, all edges get covered twice (yielding a bound of 2 on the simultaneous length approximation) and no vertex gets covered more than 2ρ times.

Using the more time-consuming method (iii), we get better approximation factors for the turn cost.

THEOREM 5.5. *There is a 1.5-approximation of complexity $O(n^3)$ for computing a minimum-turn cycle cover for a graph with n edges, and a 3.5-approximation of the same complexity for computing minimum-turn tours.*

Proof. As before, we can compute an optimal strip cover S in linear time. Analogous to the approach in section 4.1, define a weight function between endpoints of strips, taking into account the direction when leaving a strip. Clearly, any feasible tour consists of two different matchings M_1 and M_2 between strip endpoints; moreover, if $d(M_1)$ and $d(M_2)$ are the total weights of the edges in the matchings, we get $d(M_1) + d(M_2) \leq d(T)$. It follows that for an optimal matching M , we have $d(M) \leq \text{OPT}/2$. By construction, the edges of M and the strips of S induce a 2-factor of the vertices that covers all edges. Thus, we get a cycle cover of cost at most 1.5 OPT.

Now consider the c cycles in a cycle cover. c is at most the number of strips in S , which is a lower bound on the cost of an optimal tour. As the cost of merging the c cycles is at most $2c - 2$, we get a total cost of not more than 3.5 OPT. \square

5.2. Integral orthogonal. As mentioned in the preliminaries, just the pixel count N may not be a satisfactory measure for the complexity of an algorithm, as the original region may be encoded more efficiently by its boundary, and a tour may be encoded by structuring it into a small number of pieces that have a short description. It is possible to use the above ideas for approximation algorithms in this extended framework. We describe how this can be done for the integral orthogonal case, where the set of pixels is bounded by n boundary edges.

THEOREM 5.6. *There is a 10-approximation of (strongly polynomial) complexity $O(n \log n)$ for computing a minimum-turn cycle cover for a region of pixels bounded by n integral axis-parallel segments, and a 12-approximation of the same complexity for computing minimum-turn tours. In both cases, the maximum coverage of a point is at most 4, so the algorithms are also 4-approximations on length.*

For the special case in which the boundary is connected (meaning that the region has no holes), the complexities drop to $O(n)$.

Proof. The basic idea is to find a greedy rook cover, then use it to build an approximate tour. Lemma 4.12 still holds, and each strip in a star (as described in the previous section) will be a full strip. The approximation ratios follow as special cases of Theorem 5.1: In this case, $\rho = 2$ and $\delta = 4$. It remains to show how we can find a greedy rook cover in the claimed time.

Refer back to Figure 4.6. Subdivide the region by the n vertical chords through its n vertices, resulting in at most n vertical strips X_1, \dots, X_n , of widths x_1, \dots, x_n . Similarly, consider a subdivision by the n horizontal chords through the n vertices into at most n horizontal strips Y_1, \dots, Y_n , of width y_1, \dots, y_n . In total, we get a subdivision into at most n^2 cells C_{ij} . Despite this quadratic number of cells, we can deal with the overall problem in near-linear time: Note that both subdivisions can be found in time $O(n \log n)$. For the case of a connected boundary, Chazelle's linear-time triangulation algorithm [12] implies a complexity of $O(n)$.

Choose any cell C_{ij} , which is a rectangle of size $x_i \times y_j$. Then $r_{ij} = \min\{x_i, y_j\}$ rooks can be placed greedily along the diagonal of C_{ij} , without causing any interference; such a set of rooks can be encoded as one "fat" rook, described by its leftmost uppermost corner (ξ_{ij}, η_{ij}) , and its width r_{ij} . Then the strip X_i can contain at most $x_i - r_{ij}$ additional rooks, and Y_j can contain at most $y_j - r_{ij}$ rooks. Therefore, replace x_i by $x_i - r_{ij}$, and y_j by $y_j - r_{ij}$. This changes the width of at least one of the strips to zero, effectively removing it from the set of strips. After at most $2n - 1$ steps of this type, all horizontal or all vertical strips have been removed, implying that we

have a maximal greedy rook cover.

It is straightforward to see that for a fat rook at position (ξ_{ij}, η_{ij}) and width r_{ij} , there is a canonical set of r_{ij} cycles with 10 edges each that covers every pixel that can be attacked from this rook. Furthermore, there is a “fat” cycle with at most $12r_{ij} - 2$ turns that is obtained by a canonical merging of the r_{ij} small cycles. Finally, it is straightforward to merge the fat cycles. \square

If we are willing to invest more time for computation, we can find an optimal rook cover (instead of a greedy one). As discussed in the proof of Lemma 4.11, this optimal rook cover yields an optimal strip cover. An optimal strip cover can be used to get a 6-approximation, and the new running time is $O(n^{2.5} \log n)$ or $O(N^{2.376})$.

THEOREM 5.7. *There is an $O(n^{2.5} \log N)$ -time or $O(N^{2.376})$ -time algorithm that computes a milling tour with number of turns within 6 times the optimal, and with length within 4 times the optimal.*

Proof. Apply Lemma 4.11 to find an optimal strip cover of the region. (See Figure 4.6.) As described in the proof of that lemma, the cardinality of an optimal strip cover is equal to the cardinality of an optimal rook cover. As stated, the number of strips is a lower bound on the number of turns in a cycle cover or tour.

Now any strip from u to w is covered by a “doubling” cycle with edges (u, w) , (w, w) , (w, u) , (u, u) . This gives a 4-approximation to minimum-turn cycle covers. Finally apply Corollary 4.8 to get a 6-approximation to minimum-turn tours.

The claim about coverage (and hence overall length) follows from the fact that an optimal strip cover has maximum coverage 2, and hence the cycle cover has maximum coverage 4. \square

By more sophisticated merging procedures, it is possible to reduce the approximation factor for tours to a figure closer to 4. Note that in the case of N being large compared to n , the above proof grossly overestimates the cost of merging, as all cycles within a fat strip allow merging at no additional cost. However, our best approximation algorithm achieves a factor less than 4 and uses a different strategy.

THEOREM 5.8. *For an integral orthogonal polygon with n edges and N pixels, there are 2.5-approximation algorithms, with running times $O(N^{2.376} + n^3)$ and $O(n^{2.5} \log N + n^3)$, for minimum-turn cycle cover, and hence there is a polynomial-time 3.75-approximation for minimum-turn tours.*

Proof. As described in Lemma 4.11, find an optimal strip cover S , in time $O(N^{2.376})$ or $O(n^{2.5} \log N)$. Let s be its cardinality and let OPT be the cost of an optimal tour; then $\text{OPT} \geq s$.

Now consider the end vertices of the strip cover. By construction, they are part of the boundary. Because any feasible tour T must encounter each pixel and cannot cross the boundary, either any endpoint of a strip is crossed orthogonally or the tour turns at the boundary segment. In any case, a tour must have an edge that crosses an end vertex orthogonally to the strip. (Note that this edge has zero length in the case of a U-turn.)

As in section 4.1 and the proof of Theorem 5.5, define a weight function between endpoints of strips, taking into account the direction when leaving a strip. Again any feasible tour consists of two different matchings M_1 and M_2 between strip endpoints, and for an optimal matching M , we have $d(M) \leq \text{OPT}/2$.

Computing such a matching can be achieved as follows. Note that for N pixels, an optimal strip cover has $O(\min\{\sqrt{N}, n\})$ strips; by matching endpoints of neighboring strips within the same fat strip, we are left with $O(n)$ endpoints. As described in the proof of Lemma 4.11, the overall cost for computing the link distance between

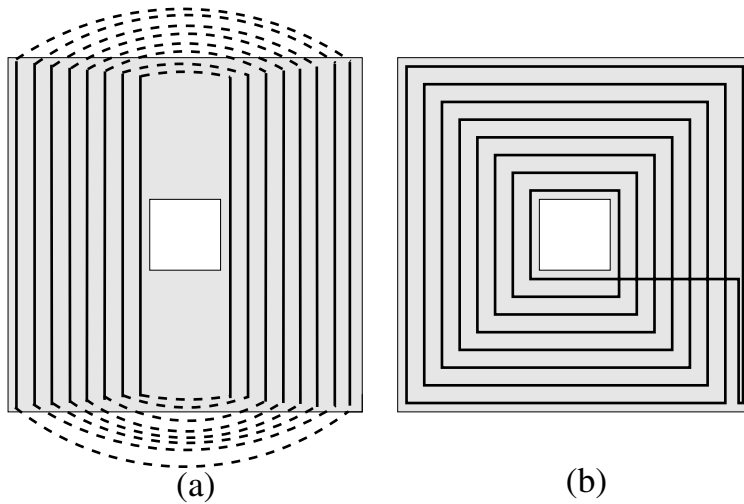


FIG. 5.2. A bad example for the 3.75-approximation algorithm: (a) Half the cycles constructed by the algorithm. (b) An optimal tour.

all pairs of endpoints can be achieved in $O(\min\{N \log N, n^2 \log n\})$. Computing a minimum-weight perfect matching can be achieved in time $O(\max\{N^{1.5}, n^3\})$.

The edges of M and the strips of S induce a 2-factor of the endpoints. Because any matching edges leave a strip orthogonally, we get at most 2 additional turns at each strip for turning each 2-factor into a cycle. The total number of turns is $2s + w(M) \leq 2.5 \cdot \text{OPT}$. Because the strips cover the whole region, we get a feasible cycle cover.

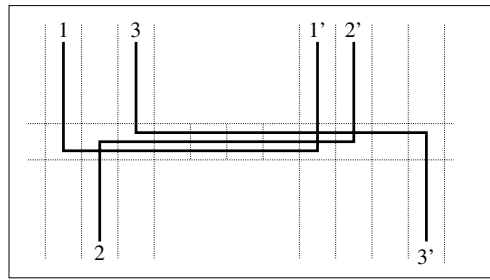
Finally, we can use Corollary 4.8 to turn the cycle cover into a tour. By the corollary, this tour does not have more than $3.75 \cdot \text{OPT}$ turns. \square

The class of examples in Example 5.9 shows that the cycle cover algorithm may use $2 \cdot \text{OPT}$ turns, and the tour algorithm may use $3 \cdot \text{OPT}$ turns, assuming that no special algorithms are used for matching and merging. Moreover, the same example shows that this 3.75-approximation algorithm does not give an immediate length bound on the resulting tour.

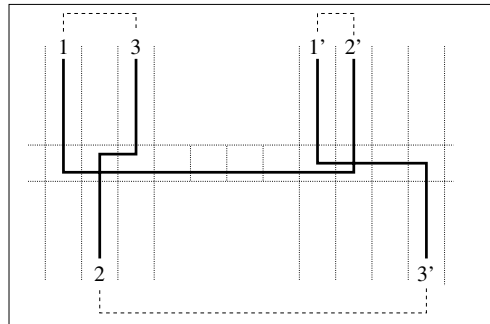
Example 5.9. The class of regions shown in Figure 5.2 may yield a heuristic cycle cover with $2 \cdot \text{OPT}$ turns, and a heuristic tour with $3 \cdot \text{OPT}$ turns.

The region consists of a “square donut” of width k . An optimal strip cover consists of $4k$ strips; an optimal matching of strip ends yields a total of $8k + 2$ turns, and we get a total of $2k$ cycles. (In Figure 5.2(a), only the vertical strips and their matching edges are shown to keep the drawing cleaner.) If the merging of these cycles is done badly (by merging cycles at crossings and not at parallel edges), it may cost another $4k - 2$ turns, for a total of $12k$ turns. As can be seen from Figure 5.2(b), there is a feasible tour that uses only $4k + 2$ turns. This shows that optimal tours may have almost all turns strictly inside of the region. Moreover, the same example shows that this 3.75-approximation algorithm does not give an immediate length bound on the resulting tour. However, we can use a local modification argument to show the following theorem.

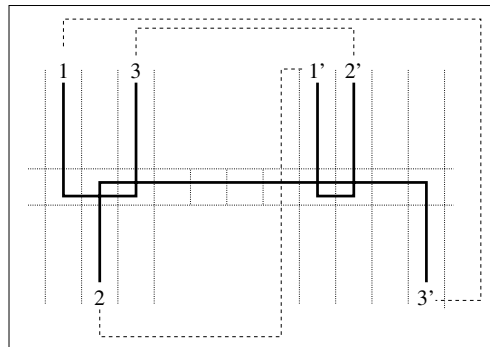
THEOREM 5.10. *For any given feasible tour (or cycle cover) of an integral orthogonal region, there is a feasible tour (or cycle cover) of equal turn number that*



(a)



(b)



(c)

FIG. 5.3. Rearranging a tour to ensure that no pixel is covered more than three times in each direction: (a) A set of horizontal edges that covers some pixel three times. (b) A rearranged tour, if the second matching between endpoints connects two left and two right endpoints. (c) A rearranged tour, if the second matching between endpoints connects any left with a right endpoint.

covers each pixel at most four times. This implies a performance ratio of 4 on the total length.

Proof. See Figure 5.3. Suppose there is a pixel that is covered at least five times. Then there is a direction (say, horizontal) in which it is covered at least three times. Let there be three horizontal segments $(1, 1')$, $(2, 2')$, $(3, 3')$ covering the same pixel, as shown in Figure 5.3(a); we denote by 1, 2, 3 the endpoints to the left of the pixel, and by $1'$, $2'$, $3'$ the endpoints to the right of the pixel.

Now consider the connections of these points by the rest of the tour, i.e., a second matching between the points 1, 2, 3, $1'$, $2'$, $3'$ that forms a cycle when merged with

the first matching $(1, 1')$, $(2, 2')$, $(3, 3')$. This second matching is shown dashed in Figure 5.3(b,c). We consider two cases, depending on the structure of the second matching.

In the first case, there are two right endpoints that are matched, say $1'$ and $2'$. Then there must be two left endpoints that are matched; because both matchings must form one large cycle, these cannot be 1 and 2. Without loss of generality, we may assume they are 1 and 3. Thus, 2 and $3'$ must be matched, as shown in Figure 5.3(b). Then we can replace $(1, 1')$, $(2, 2')$, $(3, 3')$ by $(1, 2')$, $(2, 3)$, $(1', 3')$, respectively, which yields a feasible tour with the same number of turns, but with some pixels being covered fewer times, and no pixel being covered more times than was the case in the original tour.

In the other case, all right endpoints are matched with left endpoints. Clearly, $1'$ cannot be matched with 1; without loss of generality, we assume it is matched with 2, as shown in Figure 5.3(c). Then the cycle condition implies that the second matching is $(1, 3')$, $(2, 1')$, $(3, 2')$. This allows us to replace $(1, 1')$, $(2, 2')$, $(3, 3')$ by $(1, 3)$, $(2, 3')$, $(1', 2')$, respectively, again producing a feasible tour with the same number of turns, but with some pixels being covered fewer times, and no pixel being covered more times than was the case in the original tour.

This can be repeated until no pixel is covered more than four times. As the above procedure can be carried out as part of the merging phase (i.e., after an optimal weighted matching has been found), the overall complexity is not affected. Furthermore, it is straightforward to see that it also works for the case of “thick” strips, where N is large compared to n , by treating parallel edges in a thick strip simultaneously. \square

5.3. Nonintegral orthogonal polygons. Nonintegral orthogonal polygons present a difficulty in that no polynomial-time algorithm is known to compute a minimum strip cover for such polygons. Fortunately, however, we can use the boundary tours from section 4.1 to the approximation factor of 12 from Theorem 5.6 for the integral orthogonal case.

THEOREM 5.11. *In nonintegral orthogonal milling of a polygonal region with n edges and N pixels, there is a polynomial-time 4.5-approximation for minimum-turn cycle covers and 6.25-approximation for minimum-turn tours, with a simultaneous performance guarantee of 8 on length and cover number. The running time is $O(N^{2.376} + n^3)$, or $O(n^{2.5} \log N + n^3)$.*

Proof. Take the 2.5-approximate cycle cover of the integral pixels in the region as in Theorem 5.8; for a tour, turn it into a 3.75-approximate tour. This may leave a fractional portion along the boundary uncovered. See Figure 5.4.

Now add an optimal cycle cover of the boundary which comes from Theorem 4.4. This may leave only fractional boundary pieces uncovered that are near reflex vertices of the boundary, as shown in Figure 5.4. Whenever this happens, there must be a turn of the boundary cycle cover on both sides of the reflex vertex. The fractional patch can be covered at the cost of an extra two turns, which are charged to the two turns in the boundary cycles. Therefore, the modified boundary cover has a cost of at most $2 \cdot \text{OPT}$. Compared to an optimal cycle cover of length OPT , we get a cycle cover of length at most $4.5 \cdot \text{OPT}$, as claimed. For an optimal tour of length OPT , merging all modified boundary cycles into one cycle can be done at a cost of at most 2 turns per unmodified boundary cycle, i.e., for a total of $\frac{1}{2} \cdot \text{OPT}$.

Finally, the remaining two cycles can be merged at a cost of 2 turns. This yields an overall approximation factor of $3.75 + 2.5 = 6.25$. The claim on the cover number

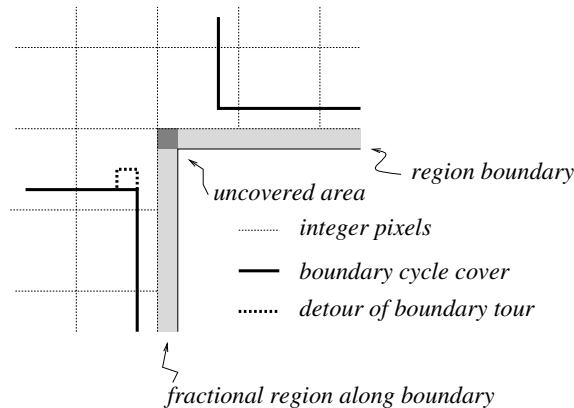


FIG. 5.4. Milling a nonintegral orthogonal polygon.

(and thus length) follows from applying Theorem 5.10 to each of the two cycles.

The running times follow from Theorems 4.4 and 5.8. \square

5.4. Milling thin orthogonal polygons. In this section we consider the special case of milling *thin* polygons. Again, we focus on the integral orthogonal case. Formally, a thin polygon is one in which no axis-aligned 2×2 square fits, implying that each pixel has all four of its corners on the boundary of the polygon. Intuitively, a polygon is thin if it is composed of a network of width-1 corridors, where each pixel is adjacent to some part of the boundary of the region, making this related to discrete milling.

5.4.1. Basics of thin orthogonal polygons. Any pixel in the polygon has one, two, three, or four neighbor pixels; we denote this number of neighbors as the degree of a pixel. See Figure 5.5. Degree-1 pixels (1) are “dead ends,” where the cutter has to make a U-turn. There are two types of degree-2 pixels, without forcing a turn (2a) or with forcing a turn (2b); in either case, applying Lemma 4.3 in an appropriate manner will suggest that they should be visited in a canonical way: after one neighbor, and before the other. Neighbors of degree-3 pixels (3) form “T” intersections that force duplication of paths. Degree-4 pixels (4) are the only pixels in thin polygons that are not boundary pixels as defined in section 4; however, in the absence of 2×2 squares of pixels, all their neighbors are of degree 1 or 2.

In the following, we will use the ideas developed for boundary cycle covers in section 4.1 to obtain cycle covers for thin polygons. The following is a straightforward consequence of Theorems 4.4 and 4.7.

COROLLARY 5.12. *In thin orthogonal milling, there is an $O(n^3)$ algorithm for computing a minimum-turn cycle cover, and an $O(n^3)$ 1.5-approximation for computing a minimum-turn tour.*

Proof. Apply the strongly polynomial algorithm described in Theorem 4.4 for computing a minimum cost boundary cycle cover. By definition, this covers all pixels of degree 1, 2, and 3. Moreover, degree-4 pixels are surrounded by pixels of degree 1 or 2, implying that they are automatically covered as neighbors of those pixels, when applying Lemma 4.3. Using Theorem 4.7, we can turn this into a tour, yielding the claimed approximation factor. \square

More interesting is that we can do much better than general merging in the case

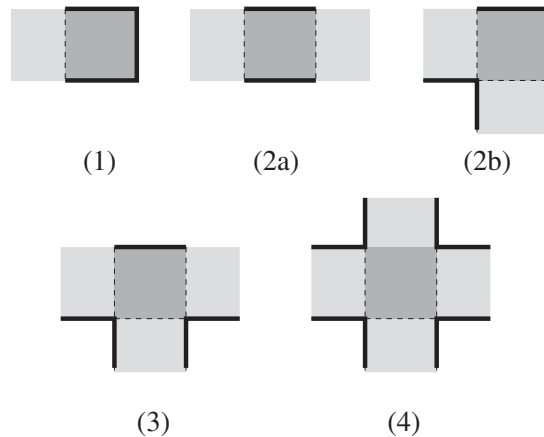


FIG. 5.5. Pixel types in a thin polygon.

of thin orthogonal milling. The idea is to decompose the induced graph into a number of cheap cycles and a number of paths.

5.4.2. Milling thin Eulerian orthogonal polygons. We first solve the special case of milling Eulerian polygons, that is, polygons that can be milled without retracing edges of the tour, so that each edge in the induced graph is traversed by the cutting tool exactly once. In an Eulerian polygon, all pixels have either two or four neighbors, meaning there are no odd-degree pixels.

Although one might expect that the optimal milling is one of the possible Eulerian tours of the graph, in fact, this is not always true, as Example 5.13 points out.

Example 5.13. *There exist thin grid graphs, such that no turn-minimal tour of the graph is an Eulerian tour.*

Proof. See Figure 5.6. Observe that an optimal milling is *not* an Eulerian tour. The best Eulerian tour for this figure requires 22 turns, as shown symbolically in the bottom left of the figure: Each cycle uses 4 turns and an additional 6 turns can be used to connect the 4 cycles together. On the other hand, the optimal milling traverses the edges in the internal pixel twice, both times in the same direction: The order of turns is 1, 2, *C*, 13, 16, 15, 14, *D*, 9, 12, 11, 10, *B*, 5, 8, 7, 6, *A*, 3, 4, 1, and the structure is shown symbolically in the bottom right. Thus, the optimal milling only requires 20 turns, where each cycle uses 4 turns and an additional 4 turns connect the cycles together. \square

By strengthening the lower bound, we can achieve the following approximation of an optimal tour of length OPT.

THEOREM 5.14. *There is an $O(n \log n)$ (or $O(N)$) algorithm that finds a tour of turn cost at most $\frac{6}{5} \cdot \text{OPT}$.*

Proof. By applying Theorem 4.4, we get an optimal boundary cycle cover. There are three observations that lead to the claimed stronger results.

(1) For a thin polygon, extracting the collinear strips can be performed in strongly polynomial time $O(n \log)$ (or weakly polynomial time $O(N)$).

(2) For an Eulerian thin polygon, no vertices in $G_{\overline{P}}$ remain unmatched after repeatedly applying Lemma 4.3. Instead, we get an optimal cycle cover right away. This cycle cover can be merged into one connecting tour by merging at pixels where two cycles cross each other: Let the optimal cycle cover be composed of c disjoint

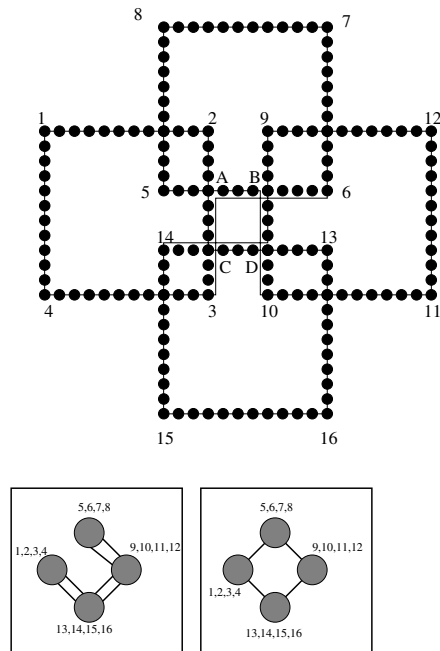


FIG. 5.6. A thin Eulerian polygon consisting of four overlapping cycles (above). Shown symbolically below is how to obtain an overall tour by merging the four canonical cycles: A tour obtained by iteratively merging cycles incurs a total of $16 + 6 = 22$ turns (bottom left). An optimal tour has $16 + 4 = 20$ turns (bottom right).

cycles, where $c \geq 1$. Let t be the cost of the optimal cycle cover. At each phase of the cycle-merging algorithm, two cycles are merged into one. Therefore, the algorithm finds a solution having cost $t + 2 \cdot (c - 1)$.

(3) We can strengthen the lower bound on an optimal tour as follows. Consider (for $c > 1$) a lower bound on the cost of the optimal solution. Just like in the proof of Theorem 3.1, all turns in a cycle cover are forced by convex corners of the polygon, implying that any solution must contain these t turns. In addition, turning from one cycle into another incurs a crossing cost of at least one turn; thus, we get a lower bound of $t + c$. Observe that there are at least 4 turns per cycle so that $t \geq 4c$. Therefore, $\frac{t+2 \cdot (c-1)}{t+c} \leq \frac{t+2c}{t+c} \leq \frac{6}{5}$. \square

5.4.3. Milling arbitrary thin orthogonal polygons. Now we consider the case of general thin polygons. For any odd-degree vertex, and any feasible solution, some edges may have to be traversed multiple times. As in Corollary 5.12, we can apply Theorem 4.4 to achieve a minimum-cost cycle cover and merge them into a tour. Using a more refined analysis, we can use this to obtain a $4/3$ -approximation algorithm for finding a minimum-cost tour.

THEOREM 5.15. *For thin orthogonal milling, we can compute a tour of turn cost at most $\frac{4}{3} \cdot \text{OPT}$ in time $O(n^3)$, where OPT is the cost of an optimal tour.*

Proof. We start by describing how to merge the cycles into one connected tour.

1. Find an optimal cycle cover as provided by Theorem 4.4.
2. Repeat until there is only one cycle in the cycle cover:
 - If there are any two cycles that can be merged without any extra cost (by having partially overlapping collinear edges), perform the merge.

- Otherwise,
 - find a vertex at which two cycles cross each other;
 - modify the vertex to incorporate at most two additional turns, thereby connecting the two cycles.

Now we analyze the performance of our algorithm. Consider the situation after extracting the cost zero matching edges from Lemma 4.3. This already yields a set K of cycles, obtained by only turning at pixels of degree 2 that force a turn. Let k denote the number of cycles in K , and let c be the number of turns in K . Let P be the set of “dangling” paths at degree-1 or degree-3 pixels, and let p be the number of turns in P , including the mandatory turn for each endpoint. Let M be a minimum matching between odd-degree vertices, and let m be the number of turns in M . Finally, let O be the matching between odd-degree pixels that is induced by an optimal tour, and let o be the number of turns in O .

First note that P is a matching between odd-degree nodes.

P , M , and O may connect some of the cycles in K . In P a path between two odd-degree pixels connects the two cycles that the two nodes belong to. On the other hand, a path in M and O between two odd-degree nodes can encounter several cycles along the way, and thus it may be used to merge several cycles at no extra cost.

Therefore, let j be the number of cycles after using P for free merging, let i be the number of components with P and M used for free merging, and let h be the number of components with P and O used for free merging.

Note that

$$(5.1) \quad 1 \leq i \leq j \leq k$$

and

$$(5.2) \quad 1 \leq h \leq j \leq k.$$

Now consider the number of cycles encountered by a path in the matching. It is not hard to see that this number cannot exceed the number of its turns. Therefore,

$$(5.3) \quad m \geq k - i \geq j - i,$$

$$(5.4) \quad o \geq k - h \geq j - h.$$

If a particular matching results in x components, we would need at least x more turns to get a tour. Thus with O we need at least h more turns.

Thus, for an optimal tour of cost OPT , we have

$$(5.5) \quad \text{OPT} \geq c + p + o + h.$$

Our heuristic method adds the minimum matching to C and P and merges the remaining components with two turns per merge, and hence the cost HEUR of the resulting tour is

$$(5.6) \quad \text{HEUR} \geq c + p + m + 2(i - 1).$$

Thus we get the following estimate for the approximation factor $R \geq 1$:

$$(5.7) \quad R \leq \frac{c + p + m + 2i}{c + p + o + h}.$$

Because each cycle has at least four turns, we know that

$$(5.8) \quad c \geq 4k \geq 4j.$$

Using the fact that $c + p + m + 2i \geq c + p + o + h$ (because $R \geq 1$), we see that the ratio on the right in (5.7) gets *larger* if we replace c in the numerator and in the denominator by the smaller nonnegative value $4j$; thus,

$$(5.9) \quad R \leq \frac{4j + p + m + 2i}{4j + p + o + h}.$$

Because P is also a matching, we have

$$(5.10) \quad p \geq m,$$

which implies that

$$(5.11) \quad R \leq \frac{4j + 2m + 2i}{4j + m + o + h}.$$

We also know that

$$(5.12) \quad m \leq o.$$

Using this, together with the fact that R can be assumed to be less than 2, we can argue that R is maximal for maximal values of m ; hence,

$$(5.13) \quad R \leq \frac{4j + 2o + 2i}{4j + 2o + h}.$$

Using (5.4) in (5.13), we see that R is maximal for minimal o ; hence,

$$(5.14) \quad R \leq \frac{4j + 2(j - h) + 2i}{4j + 2(j - h) + h} = \frac{6j - 2h + 2i}{6j - h}.$$

Using $h > o$ in (5.14) and the facts that $R < 2$ and $i \leq j$, we get that

$$(5.15) \quad R \leq \frac{6j - 2o + 2i}{6j - o} \leq \frac{6j + 2i}{6j} \leq \frac{4}{3}. \quad \square$$

The following shows that the estimate for the performance ratio is tight.

THEOREM 5.16. *There is a class of examples for which the estimate of $4/3$ for the performance ratio of the algorithm for thin orthogonal milling is tight.*

Proof. See Figure 5.7. The region consists of $k = 2s + 4$ cycles, all with precisely 4 turns, s cycles without degree-three vertices, and $s + 4$ cycles with two degree-3 vertices each. We get $c = 4k = 8s + 16$ and $p = 2(s + 4) = 2s + 8$. Figure 5.7(b) shows a min-cost matching of cost $m = 2(s + 4) = 2s + 8$ and one of cost $o = 2s + 8$ that is induced by an optimum tour. As Figure 5.7(c) suggests, merging all cycles, odd-degree paths, and matching paths is possible without requiring any further turns, resulting in $\text{OPT} = c + p + o = 12s + 32$. On the other hand, using the min-cost matching of cost m leaves k cycles that cannot be merged for free; thus, merging two cycles at a time at a cost of 2 turns requires an additional cost of $2(k - 1) = 4s + 6$, for a total of $\text{HEUR} = c + p + m + 2(k - 1) = 16s + 38$ turns, which gets arbitrarily close to $\frac{4}{3}\text{OPT}$ for large s . \square

Note that the argument of Theorem 5.10 remains valid for this section, so the bounds on coverage and length approximation still apply.

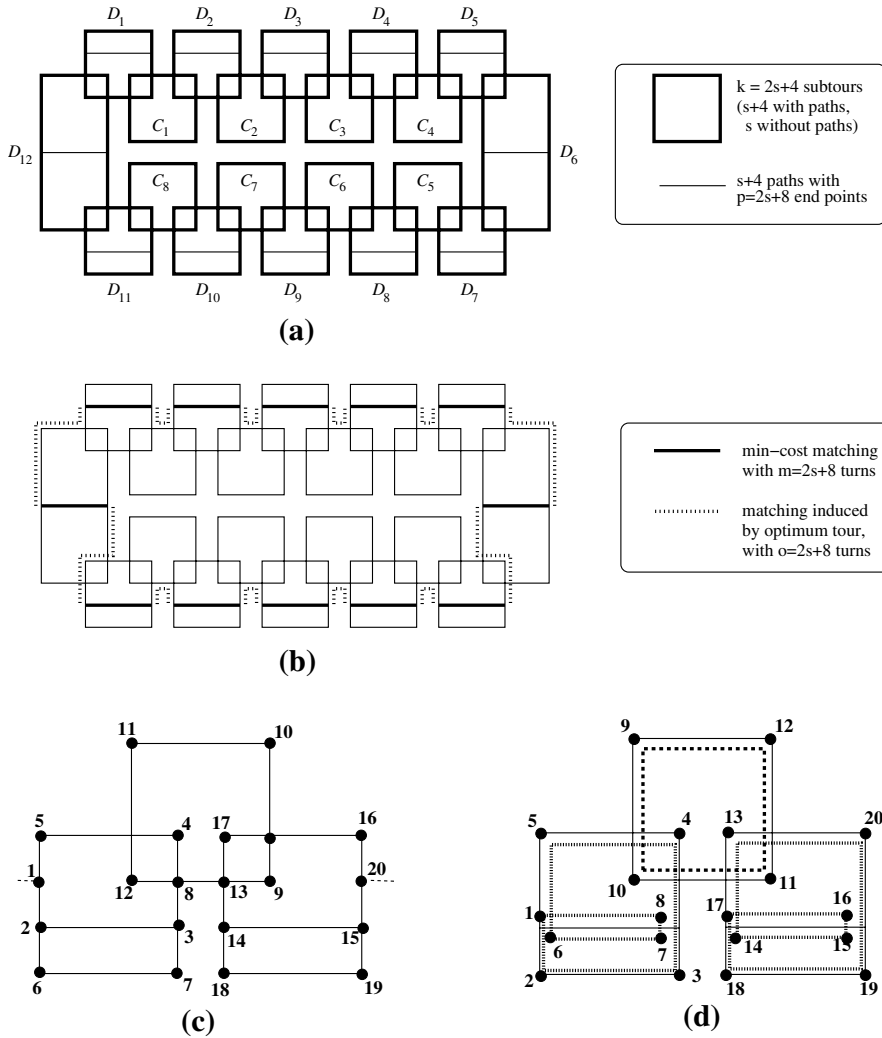


FIG. 5.7. An example with performance ratio $4/3$ for our heuristic. (a) The structure of the example for $s = 8$. (b) A min-cost matching of the odd-degree vertices, and the matching induced by an optimal tour. (c) A portion of the optimal tour: Subtours can be merged without extra cost. (d) A corresponding portion of the heuristic tour: Subtours still need to be merged, which results in an additional cost of $2(k - 1) = 4s + 6$.

5.5. PTAS. We describe a polynomial-time approximation scheme (PTAS) for the problem of minimizing a weighted average of the two cost criteria: length and number of turns. Our technique is based on using the theory of m -guillotine subdivisions [38], properly extended to handle turn costs. We prove the following result.

THEOREM 5.17. Define the cost of a tour to be its length plus C times the number of (90-degree) turns. For any fixed $\varepsilon > 0$, there is a $(1 + \varepsilon)$ -approximation algorithm, with running time $2^{O(h)} \cdot N^{O(C)}$, for minimizing the cost of a tour for an integral orthogonal polygon P with h holes and N pixels.

Proof. Let T^* be a minimum-cost tour and let m be any positive integer. Following the notation of [38], we first apply the main structure theorem of that paper to

claim that there is an m -guillotine subdivision, T_G , obtained from T^* by adding an appropriate set of bridges (m -spans, which are horizontal or vertical segments) of total length at most $\frac{1}{m}|T^*|$, with length at most $(1 + \frac{1}{m})$ times the length of T^* . (Because T^* may traverse a horizontal/vertical segment twice, we consider such segments to have multiplicities (1 or 2), as multiedges in a graph.)

We note that part of T_G may lie outside the polygon P , because the m -spans that we add to make T^* m -guillotine need not lie within P . We convert T_G into a new graph by subtracting those portions of each bridge that lie outside of P . In this way, each bridge of T_G becomes a set of segments within P ; we trim each such segment at the first and last edges of T_G that are incident on it and call the resulting trimmed segments *subbridges*. (Note that a subbridge may be a single point if the corresponding segment is incident on a single edge of T^* ; we can ignore such trivial subbridges.) As in the TSP method of [38], we *double* the (nontrivial) subbridges: We replace each subbridge by a pair of coincident segments, which we “inflate” slightly to form a degenerate loop, such that the endpoints of the subbridge become vertices of degree 4, and the endpoints of each edge incident on the interior of the subbridge become vertices of degree 3 (which occur in pairs). Refer to Figure 5.8. We let T'_G denote the resulting graph. Now $T'_G \subset P$, and, because T'_G is obtained from T^* , a tour, we know that the number of odd-degree vertices of T'_G that lie on any one subbridge is *even* (the degree-3 vertices along a subbridge come in pairs).

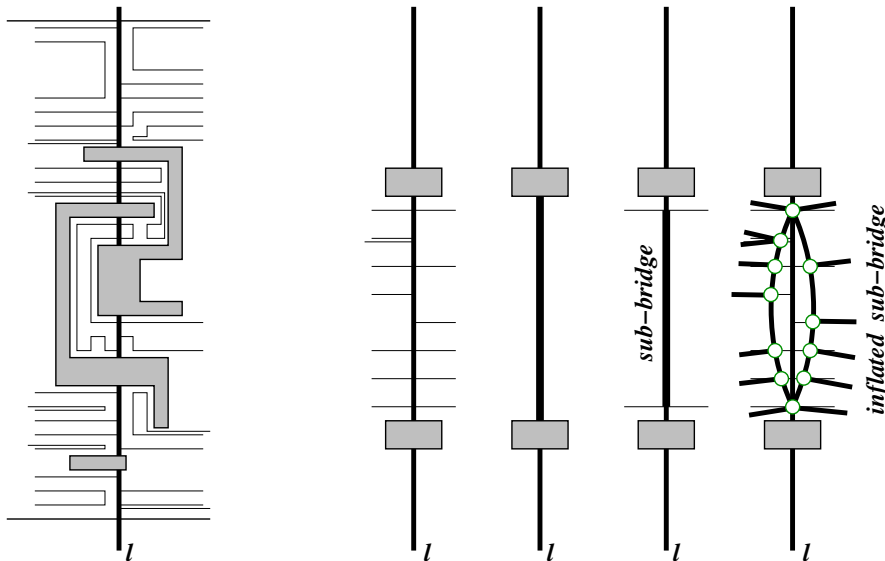


FIG. 5.8. Definitions of subbridges and the graph T'_G . A vertical line l is shown, which defines a cut. The slice of an optimal solution along l is shown, with thinner lines drawn along the edges of the solution (which is not intended to be an accurate instance of an optimal solution, but is drawn to illustrate some of the possible cases). Also shown is an enlargement of one portion of the cut l , showing a segment of the m -span between two portions of the boundary of P , the trimmed segment that forms the subbridge, and the portion of the resulting graph T'_G in the vicinity of the inflated subbridge.

The cost of the optimal solution T^* is its length, $|T^*|$, plus C times the number of its vertices. We consider the cost of T'_G to be also its (Euclidean) length plus C times the number of its vertices. Because the number of vertices on the subbridges is proportional to their total length, and each edge multiplicity is at most two, we see

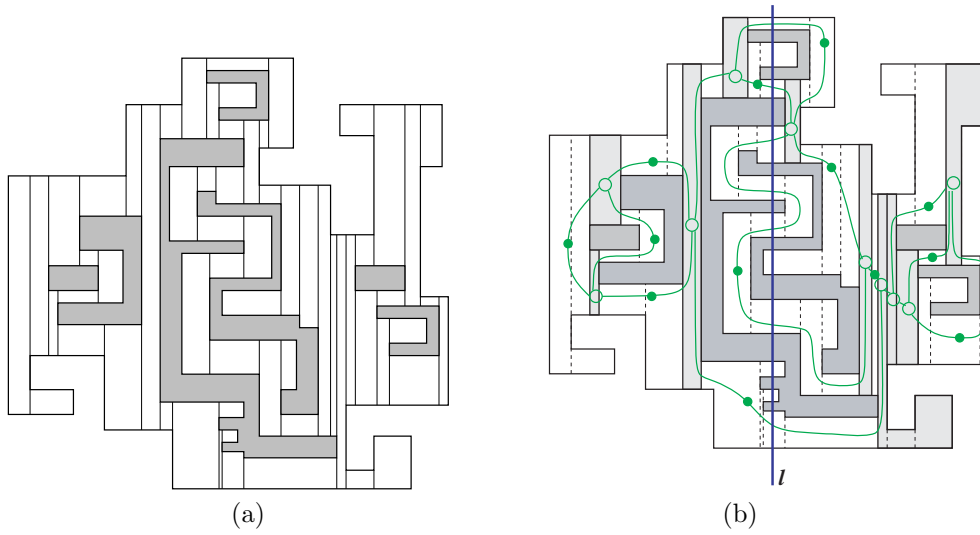


FIG. 5.9. (a) The vertical decomposition of P (holes in dark gray). (b) The decomposition of P into junctions (light gray) and corridors, with the dual graph overlaid. The nodes of the dual graph are shown as hollow for junctions and as smaller solid disks for the corridors.

that the cost of T'_G is $O((1+C)/m) \cdot |T^*|$ greater than the optimal cost, i.e., the cost of T^* .

In order to avoid exponential dependence on n in our algorithm, we need to introduce a subdivision of P that allows us to consider the subbridges along an m -span to be grouped into a relatively small ($O(h)$) number of classes. We now describe this subdivision of P .

By standard plane sweep with a vertical line, we partition P into rectangles, using vertical chords, according to the vertical decomposition; see Figure 5.9(a). We then decompose P into a set of $O(h)$ regions, each of which is either a “junction” or a “corridor.” This decomposition is analogous to the corridor structure of polygons that has been utilized in computing shortest paths and minimum-link separators (see, e.g., [32, 33, 40]), with the primary difference being that we use the vertical decomposition into rectangles, rather than a triangulation, as the basis of the definition. Consider the (planar) dual graph, \mathcal{G} , of the vertical partition of P ; the nodes of \mathcal{G} are the rectangles, and two nodes are joined by an edge if and only if they are adjacent. We now define a process to transform the vertical decomposition into our desired decomposition. First, we take any degree-1 node of \mathcal{G} and delete it, along with its incident edge; in the vertical decomposition, we remove the corresponding vertical chord (dual to the edge of \mathcal{G} that was deleted). We repeat this process, merging a degree-1 region with its neighbor, until there are no degree-1 nodes in \mathcal{G} . At this stage, \mathcal{G} has $h+1$ faces and all nodes are of degree 2 or more. Assume that $h \geq 2$ (the case $h \leq 1$ is easy); then not all nodes are of degree 2, implying that there are at least two higher-degree nodes. Next, for each pair of adjacent degree-2 nodes, we merge the nodes, deleting the edge between them and removing the corresponding vertical chord separating them in the decomposition. The final dual graph \mathcal{G} has nodes of two types: those that are dual to regions of degree 2, which we call the *corridors*, and those that are dual to regions of degree greater than 2, which we call the *junctions*. Each corridor is bounded by exactly two vertical chords, together with two portions of

the boundary of P . (These two portions may, in fact, come from the same connected component of the boundary of P .) Each of the h bounded faces of \mathcal{G} contains exactly one of the holes of P . Refer to Figure 5.9.

Let \mathcal{V} denote the decomposition of P just described; there is an analogous horizontal partition, \mathcal{H} , of P into $O(h)$ regions. The vertical subbridges of a vertical bridge are partitioned into $O(h)$ classes according to the identity of the region, τ , that contains the subbridge in the vertical decomposition \mathcal{V} . A subbridge intersecting a region τ of \mathcal{V} is called *separating* if it separates some pair of vertical chords on the boundary of τ ; it is *trivial* otherwise. (Corridor regions have only two vertical chords on their boundary, while junctions may have several, as many as $\Theta(h)$ in degenerate cases.)

First, consider a corridor region τ in \mathcal{V} , and let a and b denote the two vertical chords that bound it. An important observation regarding subbridge classes in corridors is this: The parity of the number of times a tour crosses a must be the same as the parity of the number of times a tour crosses b . The consequence is that we can specify the parity of the number of incidences on *all* separating subbridges of a given corridor class by specifying the parity of the number of incidences on a single subbridge of the class; the trivial subbridges always have an even parity of crossing.

Now consider a junction region τ in \mathcal{V} . Because, in the merging process that defines \mathcal{V} , we never merge a degree-2 region with a higher-degree region, we know that τ consists of a single high-degree (> 2) rectangle, R_τ , from the original vertical decomposition, together with possibly many other rectangles that form a “pocket” attached to R_τ , corresponding to a tree in the dual graph (so that removal of degree-1 nodes leads to a collapse of the tree and a merging of the pocket to R_τ). The consequence of this observation is that there can be at most one separating subbridge in a junction class. (There may be several trivial subbridges.)

Our algorithm applies dynamic programming to obtain a minimum-cost m -guillotine subdivision, T_G^* , from among all those m -guillotine subdivisions that have the following additional properties:

- (1) It consists of a union of horizontal/vertical segments, having half-integral coordinates, within P .
- (2) It is connected.
- (3) It *covers* P , in that the center of every pixel of P is intersected by an edge of the subdivision.
- (4) It is a *bridge-doubled* m -guillotine subdivision, so that every (nontrivial) subbridge of an m -span appears twice (as a multiedge).
- (5) It interconnects the subbridges in each of a specified partition of the classes of subbridges.
- (6) It obeys a parity constraint on each of the $O(h)$ classes of subbridges: The number of edges of the subdivision incident on each separating subbridge of the class corresponding to a region τ is even or odd, according to the specified parity for τ .

A subproblem in the dynamic programming algorithm is specified by a rectangle, having half-integral coordinates, together with *boundary information* associated with the rectangle, which specifies how the subdivision within the rectangle interacts with the subdivision outside the rectangle. The boundary information includes (a) $O(m)$ attachment points, where edges meet the boundary at points other than along the m -span; (b) the multiplicity (1 or 2) of each attachment point, and the interconnection pattern (if any) of adjacent attachment points along the rectangle boundary; (c) the endpoints of a bridge on each side of the rectangle (from which one can deduce

the subbridges); (d) interconnection requirements among the attachment points and the classes of subbridges; and (e) a parity bit for each class of subbridge, indicating whether an even or an odd number of edges should be incident to the separating subbridges of that class. There are $N^{O(m)} \cdot 2^{O(h)}$ subproblems. At the base of the dynamic programming recursion are rectangles of constant size (e.g., unit squares).

The optimization considers each possible cut (horizontal or vertical, at half-integral coordinates) for a given subproblem, together with all possible choices for the new boundary information along the cut, and minimizes the total resulting cost, adding the costs of the two resulting subproblems to the cost of the choices made at the cut (which includes the length of edges added, plus C times the number of vertices added).

Once an optimal subdivision, T_G^* , is computed, we can recover a valid tour from it by extracting an Eulerian subgraph obtained by removing a subset of the edges on each doubled subbridge. The parity conditions imply that such an Eulerian subgraph exists. An Eulerian tour on this subgraph is a covering tour, and its cost is at most $O(C/m) \cdot |T^*|$ greater than optimal. For any fixed $\varepsilon > 0$, we set $m = \lceil C/\varepsilon \rceil$, resulting in a $(1 + \varepsilon)$ -approximation algorithm with running time $O(N^{O(C/\varepsilon)} \cdot 2^{O(h)})$. The techniques of [38], which use “grid-rounded” m -guillotine subdivisions, can be applied to reduce the exponent on N to a term, $O(C)$, independent of m . \square

Remarks. It should be possible to apply a variant of our methods to obtain a PTAS that is polynomial in n (versus N), with a careful consideration of implicit encodings of tours. We note that our result relies on effectively “charging off” turn cost to path length, because the objective function is a linear combination of the two costs (turns and length). We have not yet been able to give a PTAS for minimizing only the number of turns in a covering tour; this remains an intriguing open problem.

6. Conclusion. We have presented a variety of results for finding an optimal tour with turn cost. Many open problems remain. The most curious seems to be the following, which highlights the difference between turn cost and length, as well as the difference between a cycle cover and a 2-factor.

PROBLEM 6.1. *What is the complexity of finding a minimum-turn cycle cover in a grid graph?*

This problem has been open for several years now; in fact, it is Problem # 53 on the well-known list [17], known as “The Open Problems Project.” Finding a minimum weighted turn cycle cover is known to be NP-hard for a set of points in the plane [1]; however, the proof uses the fact that there are more than two directions for the edges in a convex cycle. While we tend to believe that Problem 6.1 may have the answer “NP-complete,” a polynomial solution would immediately lead to a 1.5-approximation for the orthogonal case, and a $(1 + \frac{2}{3})$ -approximation for the general case.

For various optimization problems dealing with geometric regions, there is a notable difference in complexity between a region with holes and a *simple* region that does not have any holes. (In particular, it can be decided in polynomial time whether a given grid graph without holes has a Hamiltonian cycle [44], even though the complexity of the TSP on these graphs is still open.) Our NP-hardness proof makes strong use of holes; furthermore, the complexity of the PTAS described above is exponential in the number of holes. This raises the following natural question.

PROBLEM 6.2. *Is there a polynomial-time algorithm for exactly computing a minimum-turn covering tour for simple orthogonal polygons?*

It may be possible to improve the performance of some of our approximation algorithms. In particular, the following remains unclear.

PROBLEM 6.3. *Is the analysis of the 3.75-approximation algorithm tight?*

We believe that it may be possible to improve the factor. We also believe that there is room for improvement in approximating nonintegral orthogonal milling, in particular by improving the cost of finding a strip cover.

PROBLEM 6.4. *What is the complexity of computing a minimum strip cover in nonintegral orthogonal polygons?*

An important tool for our approximation algorithms is a strip cover of small cost; finding a strip cover remains a possible approach even if strips may be parallel to more than two directions. This is closely related to other decomposition problems; see [34] for a survey.

PROBLEM 6.5. *What is the complexity of computing minimum strip covers in nonorthogonal polygons?*

The answer may very well be “NP-hard, even for three directions”: Hassin and Megiddo [26] show that the problem of hitting a set of points with a minimum number of lines with three slopes is hard. However, their proof constructs a disconnected set of grid points and cannot be applied directly to milling problems. In any case, even an approximation would be of interest, in particular if it achieves the following property.

PROBLEM 6.6. *Is there a strip cover approximation algorithm for d directions whose performance ratio is independent of d ?*

This would imply a positive result for a special case of the following, even more general, problem.

PROBLEM 6.7. *Can one obtain approximation algorithms for unrestricted directions in an arbitrary polygonal domain, and an appropriately shaped cutter?*

Acknowledgments. We are obliged to Valentin Polishchuk for a very thorough list of suggestions, and thank three anonymous referees for various comments that helped to improve the presentation of the paper. We thank Regina Estkowski for helpful discussions.

REFERENCES

- [1] A. AGGARWAL, D. COPPERSMITH, S. KHANNA, R. MOTWANI, AND B. SCHIEBER, *The angular-metric traveling salesman problem*, in Proceedings of the 8th Annual ACM-SIAM Symposium on Discrete Algorithms, SIAM, Philadelphia, 1997, pp. 221–229.
- [2] M. H. ALSUWAIYEL AND D. T. LEE, *Minimal link visibility paths inside a simple polygon*, *Comput. Geom. Theory Appl.*, 3 (1993), pp. 1–25.
- [3] M. H. ALSUWAIYEL AND D. T. LEE, *Finding an approximate minimum-link visibility path inside a simple polygon*, *Inform. Process. Lett.*, 55 (1995), pp. 75–79.
- [4] E. M. ARKIN, M. A. BENDER, E. D. DEMAINE, S. P. FEKETE, J. S. B. MITCHELL, AND S. SETHIA, *Optimal covering tours with turn costs*, in Proceedings of the 12th Annual ACM-SIAM Symposium on Discrete Algorithms, SIAM, Philadelphia, 2001, pp. 138–147.
- [5] E. M. ARKIN, S. P. FEKETE, AND J. S. B. MITCHELL, *The lawnmower problem*, in Proceedings of the 5th Canadian Conference on Computational Geometry, University of Waterloo, Waterloo, ON, Canada, 1993, pp. 461–466.
- [6] E. M. ARKIN, S. P. FEKETE, AND J. S. B. MITCHELL, *Approximation algorithms for lawn mowing and milling*, *Comput. Geom. Theory Appl.*, 17 (2000), pp. 25–50.
- [7] E. M. ARKIN, M. HELD, AND C. L. SMITH, *Optimization problems related to zigzag pocket machining*, *Algorithmica*, 26 (2000), pp. 197–236.
- [8] E. M. ARKIN, J. S. B. MITCHELL, AND C. D. PIATKO, *Minimum-link watchman tours*, *Inform. Process. Lett.*, 86 (2003), pp. 203–207.
- [9] S. ARYA, S.-W. CHENG, AND D. M. MOUNT, *Approximation algorithm for multiple-tool milling*, *Internat. J. Comput. Geom. Appl.*, 11 (2001), pp. 339–372.
- [10] A. A. ASSAD AND B. L. GOLDEN, *Arc routing methods and applications*, in Network Routing, M. O. Ball, T. L. Magnanti, C. L. Monma, and G. L. Nemhauser, eds., *Handbooks Oper. Res. Management Sci.* 8, Elsevier Science, Amsterdam, 1995, pp. 375–483.

- [11] E. BENAVENT AND D. SOLER, *The directed rural postman problem with turn penalties*, *Transportation Sci.*, 33 (1999), pp. 408–418.
- [12] B. CHAZELLE, *Triangulating a simple polygon in linear time*, *Discrete Comput. Geom.*, 6 (1991), pp. 485–524.
- [13] J. CLOSSEY, G. LAPORTE, AND P. SORIANO, *Solving Arc Routing Problems with Turn Penalties*, Technical report G-2000-05, Le Groupe d'études et de recherche en analyse des décisions (GERAD), Montréal, Canada, 2000.
- [14] M. J. COLLINS AND B. M. E. MORET, *Improved lower bounds for the link length of rectilinear spanning paths in grids*, *Inform. Process. Lett.*, 68 (1998), pp. 317–319.
- [15] J. CULBERSON AND R. A. RECKHOW, *Orthogonally convex coverings of orthogonal polygons without holes*, *J. Comput. System Sci.*, 39 (1989), pp. 166–204.
- [16] E. D. DEMAINE, M. L. DEMAINE, AND J. S. B. MITCHELL, *Folding flat silhouettes and wrapping polyhedral packages: New results in computational origami*, in *Proceedings of the 15th Annual ACM Symposium on Computational Geometry*, 1999, pp. 105–114.
- [17] E. D. DEMAINE, J. S. B. MITCHELL, AND J. O'ROURKE, *The Open Problems Project*, <http://maven.smith.edu/~orourke/TOPP/>.
- [18] H. A. EISELT, M. GENDREAU, AND G. LAPORTE, *Arc routing problems, part I: The Chinese postman problem*, *Oper. Res.*, 43 (1995), pp. 231–242.
- [19] H. A. EISELT, M. GENDREAU, AND G. LAPORTE, *Arc routing problems, part II: The rural postman problem*, *Oper. Res.*, 43 (1995), pp. 399–414.
- [20] S. P. FEKETE, *Geometry and the Travelling Salesman Problem*, Ph.D. thesis, Department of Combinatorics and Optimization, University of Waterloo, Waterloo, ON, Canada, 1992.
- [21] S. P. FEKETE AND G. J. WOEGINGER, *Angle-restricted tours in the plane*, *Comput. Geom. Theory Appl.*, 8 (1997), pp. 195–218.
- [22] D. S. FERNÁNDEZ, *Problemas de Rutas por Arcos con Giros Prohibidos*, Ph.D. thesis, Vniversitat de Valencia, Valencia, Spain, 1995.
- [23] H. N. GABOW AND R. E. TARJAN, *Faster scaling algorithms for network problems*, *SIAM J. Comput.*, 18 (1989), pp. 1013–1036.
- [24] H. N. GABOW, *Implementation of Algorithms for Maximum Matching on Nonbipartite Graphs*, Ph.D. thesis, Department of Computer Science, Stanford University, Stanford, CA, 1973.
- [25] M. R. GAREY AND D. S. JOHNSON, *Computers and Intractability: A Guide to the Theory of NP-Completeness*, W. H. Freeman, New York, 1979.
- [26] R. HASSIN AND N. MEGIDDO, *Approximation algorithms for hitting objects by straight lines*, *Discrete Appl. Math.*, 30 (1991), pp. 29–42.
- [27] M. HELD, *On the Computational Geometry of Pocket Machining*, *Lecture Notes in Comput. Sci.* 500, Springer-Verlag, Berlin, 1991.
- [28] O. H. IBARRA AND S. MORAN, *Deterministic and probabilistic algorithms for maximum bipartite matching via fast matrix multiplication*, *Inform. Process. Lett.*, 13 (1981), pp. 12–15.
- [29] A. ITAI, C. H. PAPADIMITRIOU, AND J. L. SZWARCFITER, *Hamilton paths in grid graphs*, *SIAM J. Comput.*, 11 (1982), pp. 676–686.
- [30] K. IWANO, P. RAGHAVAN, AND H. TAMAKI, *The traveling cameraman problem, with applications to automatic optical inspection*, in *Proceedings of the 5th Annual International Symposium on Algorithms and Computation*, *Lecture Notes Comput. Sci.* 834, Springer-Verlag, Berlin, 1994, pp. 29–37.
- [31] D. S. JOHNSON AND C. H. PAPADIMITRIOU, *Computational complexity and the traveling salesman problem*, in *The Traveling Salesman Problem*, E. Lawler, J. Lenstra, A. R. Kan, and D. Shmoys, eds., John Wiley and Sons, New York, 1985, pp. 68–74.
- [32] S. KAPOOR AND S. N. MAHESHWARI, *Efficient algorithms for Euclidean shortest path and visibility problems with polygonal obstacles*, in *Proceedings of the 4th Annual ACM Symposium on Computer Geometrics*, 1988, pp. 172–182.
- [33] S. KAPOOR, S. N. MAHESHWARI, AND J. S. B. MITCHELL, *An efficient algorithm for Euclidean shortest paths among polygonal obstacles in the plane*, *Discrete Comput. Geom.*, 18 (1997), pp. 377–383.
- [34] J. M. KEIL, *Polygon decomposition*, in *Handbook of Computational Geometry*, J.-R. Sack and J. Urrutia, eds., Elsevier Science, North-Holland, Amsterdam, 2000, pp. 491–518.
- [35] R. KLEIN, *Personal communication*, 2000.
- [36] E. KRANAKIS, D. KRIZANC, AND L. MEERTENS, *Link length of rectilinear Hamiltonian tours on grids*, *Ars Combinatorica*, 38 (1994), p. 177.
- [37] J. S. B. MITCHELL, *L_1 shortest paths among polygonal obstacles in the plane*, *Algorithmica*, 8 (1992), pp. 55–88.
- [38] J. S. B. MITCHELL, *Guillotine subdivisions approximate polygonal subdivisions: A simple polynomial-time approximation scheme for geometric TSP, k-MST, and related problems*, *SIAM J. Comput.*, 28 (1999), pp. 1298–1309.

- [39] J. S. B. MITCHELL, *Geometric shortest paths and network optimization*, in Handbook of Computational Geometry, J.-R. Sack and J. Urrutia, eds., Elsevier Science, North-Holland, Amsterdam, 2000, pp. 633–701.
- [40] J. S. B. MITCHELL AND S. SURI, *Separation and approximation of polyhedral objects*, Comput. Geom. Theory Appl., 5 (1995), pp. 95–114.
- [41] E. M. MOLADA AND D. S. FERNÁNDEZ, *Exact solution of the Chinese postman problem with turn penalties*, in Proceedings of the 15th Congress on Differential Equations and Applications and Congress on Applied Mathematics, Vols. I and II, Colecc. Congr. 9, Servicio de Publicaciones da Universidade de Vigo, Vigo, Spain, 1998, pp. 1111–1115 (in Spanish).
- [42] S. NTAFOU, *Watchman routes under limited visibility*, Comput. Geom. Theory Appl., 1 (1992), pp. 149–170.
- [43] A. SCHRIJVER, *Combinatorial Optimization*, Springer-Verlag, Berlin, 2003.
- [44] C. UMANS AND W. LENHART, *Hamiltonian cycles in solid grid graphs*, in Proceedings of the 38th Annual IEEE Symposium on Foundations of Computer Science, 1997, pp. 496–507.
- [45] D. WEST, *Introduction to Graph Theory*, Prentice-Hall, Upper Saddle River, NJ, 1996.

ON THE COMPLEXITY OF NETWORK SYNCHRONIZATION*

DARIN GOLDSTEIN[†] AND KOJIRO KOBAYASHI[‡]

Abstract. We show that if a minimal-time solution to a fundamental distributed computation primitive, synchronizing a network path of finite-state processors, exists on the three-dimensional, undirected grid, then we can conclude the purely complexity-theoretic result $P = NP$.

Every previous result on network synchronization for various network topologies either demonstrates the existence of fast synchronization solutions or proves that a synchronization solution cannot exist at all. To date, it is unknown whether there is a network topology for which there exists a synchronization solution but for which no *minimal-time* synchronization solution exists. Under the assumption that $P \neq NP$, this paper solves this longstanding open problem in the affirmative.

Key words. firing squad synchronization problem, network synchronization, distributed computation

AMS subject classifications. 68Q25, 68W40

DOI. 10.1137/S0097539705447086

1. Introduction. The firing squad synchronization problem (FSSP) is a famous problem originally posed almost half a century ago. A prisoner is about to be executed by firing squad. The firing squad is made up of soldiers who have formed in a straight line with muskets aimed at the prisoner. Some of the soldiers are using blanks and some are using real bullets. There are good reasons for this: Nobody will ever know who the true executioner of the prisoner is, and no single member of the firing squad is able to save the condemned man's life by not firing, which reduces the moral compulsion not to fire when the order is given. But for these same reasons, it is absolutely imperative that all of the soldiers fire simultaneously.

The general stands on the left side of the line, ready to give the order, but he knows that he can only communicate with the soldier to his right. In fact, each soldier can only communicate with the soldier to his immediate left and/or right but nobody else. Soldiers have limited memory and can only pass along simple instructions. Is it possible to come up with a protocol, independent of the size of the line, for getting all of the soldiers to fire at the prisoner simultaneously if their only means of communication are small, whispered instructions only to adjacent soldiers? (The possibility of counting the number of soldiers in the line can be discounted because no soldier can remember such a potentially large amount of information; each soldier can only remember messages that are independent of the size of the line.)

The problem itself is interesting as a mathematical puzzle. More importantly, there are also applications to the synchronization of small, fast processors in large networks. In the literature on the subject (e.g., [29, 18]), the problem has been referred to as “macrosynchronization given microsynchronization” and “realizing global

*Received by the editors February 18, 2005; accepted for publication (in revised form) August 30, 2005; published electronically December 8, 2005. A preliminary version of this paper appeared in *Proceedings of the 15th International Symposium on Algorithms and Computation*, Hong Kong, China, 2004, Lecture Notes in Comput. Sci. 3341, R. Fleischer and G. Trippen, eds., Springer-Verlag, Berlin, 2004, pp. 496–507.

<http://www.siam.org/journals/sicomp/35-3/44708.html>

[†]Department of Computer Engineering and Computer Science, California State University, Long Beach, CA 90840 (daring@cecs.csulb.edu).

[‡]Department of Information Systems Science, Faculty of Engineering, Soka University, Tokyo 192-8577, Japan (kobayasi@t.soka.ac.jp).

synchronization using only local information exchange.” The synchronization of multiple small but fast processors in general networks is a fundamental problem of parallel processing and a computing primitive of distributed computation.

1.1. Early history. The FSSP has a rich history; solutions to various subproblems have been discovered over a period of decades. We summarize the history here. J. Myhill introduced the problem in 1957, though the first published reference is [26] from 1962. McCarthy and Minsky first solved the problem for the bidirectional line (as described above) in [25]. In an unpublished 1962 manuscript, Goto [11] also solved the problem on the bidirectional line. Goto’s solution fired the bidirectional line in time $2n - 2$, where n is the number of soldiers in the line; $2n - 2$ is equal to the time it would take a message to percolate from the general to the furthest soldier and back again and is therefore obviously minimal-time. Goto’s solution required a great number of states, but researchers have since reduced the number to six over a period of decades (culminating with Mazoyer’s solution in [22]). As the number of states required per automata to perform the computation has decreased from thousands to merely six, the allowable speed of the process has correspondingly increased; in general, smaller processors have higher clock speeds.¹ Given that processor synchronization is a fundamental primitive of distributed computation, it is an important problem to find solutions that are both theoretically minimal-time (i.e., requiring minimal-time steps) and as small as possible (i.e., allowing for higher clock speed/lower actual time per step).

1.2. The model. As mentioned previously, we wish to model the operation of a large network of processors whose computations are all governed synchronously by the same global clock. The model is intended to mathematically abstract a physical switching network or a very large-scale parallel processing machine. The processors are designed to be small, fast, and unable to access large memory caches. Each processor is identical and assumed to have a fixed constant number of ports which can both send and receive a constant amount of data per clock cycle. (“Constant” quantities must be independent of the size and topology of the network.) One processor is specified as the root. Its purpose is to begin the algorithm. (Intuitively, the root has the job of the “general.”)

More formally, the problem is to construct a deterministic finite-state automaton with a transition function that satisfies certain conditions. We assume that each processor in the network is identical. Initially, the root is in a special “general” state. All other nodes are initially in a special “quiescent” state, in which, at each time step, the processor sends a “blank” character through all of its ports. A processor remains in the quiescent state until a nonblank character is received by one of its inports. We consider connected networks of such identical synchronous finite-state automata with vertex degree uniformly bounded by a constant.² These automata are meant to model very small, fast processors. The network itself may have a specific topology (see below) but potentially unbounded size. The network is formed by connecting ports of a given automaton to those of other automata with wires. Not all ports of a given automaton need necessarily be connected to other automata. The network has a global clock, the pulses between which each processor performs its computations.

¹Note that the number of clock cycles required by the algorithm does not necessarily decrease. It is the time taken by *each* clock cycle that decreases.

²The vertex degree of each processor must be bounded uniformly by a constant; otherwise, a constant-sized automaton might not be able to distinguish its own ports.

Processors synchronously, within a single global clock pulse, perform the following actions in order: read in the inputs from each of their ports, process their individual state changes, and prepare and broadcast their outputs. As mentioned, our network structure is specifically designed to model the practical situation of many small and fast processors performing a synchronous distributed computation. The goal of the protocol is to cause every process in the network to enter the same special “firing” state for the first time *simultaneously*. In keeping with current technological trends that are likely to continue indefinitely, we assume that the time it takes for messages to travel along the various wires is vastly greater than the time for a simple processor state change. Our global clock pulses are therefore used to count the number of global “message interchanges” that occur.

A *solution* A for a given network topology (e.g., the bidirectional line, as above) is defined to be the instantiation of an automaton with a transition function that satisfies the firing conditions outlined above for any network size. (So, by this definition, a *solution* for the bidirectional line must function for a bidirectional line of *any* size.) Assuming a solution A is specified, the *firing time* of A on a given network will refer to the number of clock cycles it takes for this network of processors programmed with the solution A to complete the protocol and simultaneously fire. The *minimum firing time* of a given network (of a specified topology) will refer to the minimum over all solutions A of the firing time of the network of processors programmed with solution A . A *minimal-time solution* A_{min} for a given network topology will be a solution such that the firing time for a network of any given size (with the given topology) programmed with the algorithm A_{min} will equal the minimum firing time of the network. Note that even though the network can be of arbitrary size, the size of the algorithm A_{min} must be fixed.

1.3. Additional history. Numerous other variations of the problem have also been studied, and there is a large body of results that spans almost 50 years. Mazoyer provides an overview of the problem (up to 1986, at least) in addition to some of its history in [23]. We give a brief survey of results and open problems restricting ourselves only to variations that satisfy the following conditions.

- ◇ The structure of the configurations (placements of soldiers) that we consider will not change dynamically. Once the protocol is initiated, the network is considered static.
- ◇ Adjacent soldiers can exchange information with one unit time delay. (Automata-theoretically speaking, the state of a soldier at time t depends only on the states of the soldier and adjacent soldiers at time $t - 1$.)
- ◇ There are no restrictions on how much information is exchanged between adjacent soldiers. For example, there are variations of the problem that restrict the messages to be only a single bit long. Because the automata under consideration are finite-state, messages are restricted to be of constant size but not necessarily just a single bit.

1.3.1. Variations for which minimal-time solutions are known. The following lists some of the variations for which minimal-time solutions to the FSSP are known and their corresponding minimum firing times.

- The original FSSP on the bidirectional line of length n : The minimum firing time is $2n - 2$ (see Goto [11], Waksman [38], Balzer [1], and Mazoyer [22]).
- The one-dimensional line of length n such that the position of the root may be at any position: The minimum firing time is $2n - 2 - \min\{k - 1, n - k\}$, where k is the position of the root ($1 \leq k \leq n$) (see Moore and Langdon [27],

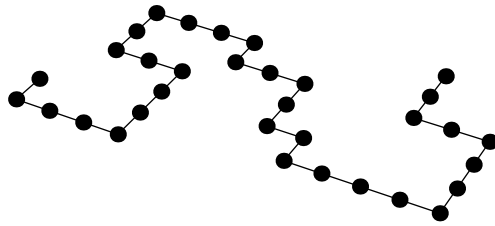


FIG. 1.1. An example of a 2PATH configuration.

Varshavsky, Marakhovsky, and Peschansky [37], Szwerinski [36], and Settle and Simon [34]).

- The square of size $n \times n$: The minimum firing time is $2n - 2$ (see Shinahr [35]).
- The rectangle of size $m \times n$: The minimum firing time is $m + n + \max\{m, n\} - 3$ (see Shinahr [35]).
- The cube of size $n \times n \times n$: The minimum firing time is $3n - 3$ (see Shinahr [35]).
- The bilateral ring of size n : The minimum firing time is n (see Culik [5] and Berthiaume et al. [3]).
- The ring of size n with one-way information flow: The minimum firing time is $2n - 1$ (see Kobayashi [20] and LaTorre, Napoli, and Parente [21]).

1.3.2. Variations for which minimal-time solutions are not known. The following lists five basic variations for which solutions have been shown to exist but for which no minimal-time solutions are known.

- FSSP for paths in the two-dimensional or the three-dimensional grid spaces: A configuration is a path in the two-dimensional or the three-dimensional grid spaces, and the position of the root is at one of the two endpoints. We abbreviate these problems 2PATH and 3PATH, respectively. In Figure 1.1 we show an example of such configurations for 2PATH. These are the natural generalizations of the original FSSP.
- FSSP for paths in the two-dimensional or the three-dimensional grid spaces such that the position of the root may be at any position in the path: We call these problems the generalized FSSP for paths in the two-dimensional or the three-dimensional grid spaces, respectively, and abbreviate them g-2PATH and g-3PATH. These problems are natural generalizations of the variation studied by Moore and Langdon [27].
- FSSP for regions in the two-dimensional or the three-dimensional grid spaces: A configuration is a connected finite subset of grid points in the two-dimensional or the three-dimensional grid spaces. We abbreviate these problems 2REG and 3REG, respectively. These problems are natural generalizations of squares, rectangles, and cubes. In Figure 1.2, we show an example of a 2REG configuration. We discuss the 2REG, a subproblem of FSSP for general undirected networks, in greater detail below. The first solution by Rosenstiehl [31] and Rosenstiehl, Fiksel, and Holliger [32] for the latter FSSP was also the first solution for 2REG. Romani [30] improved that solution, and this improvement also applied to 2REG. At present, the best solution by

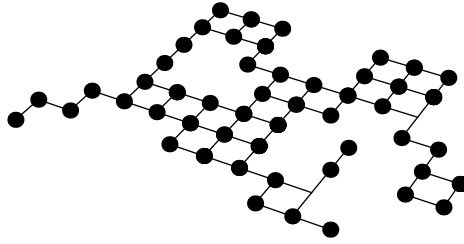


FIG. 1.2. An example of a 2REG configuration.

Nishitani and Honda [28] for FSSP for general undirected networks mentioned below is also the best solution for 2REG. Some fast solutions that work only for some special types of two-dimensional regions are also known (see Grasselli [12] and Kobayashi [15]).

- FSSP for general undirected networks:
Each soldier is modeled by a finite automaton having a constant number of bidirectional terminals. The study of this version of the FSSP started with Rosenstiehl [31] and Rosenstiehl, Fiksel, and Holliger [32]. At present, the solution with the smallest firing time is by Nishitani and Honda [28]. The firing time of their solution is $3r - 1$, where r is the radius of the network. The coefficient 3 is optimal in the sense that there is no solution having firing time $cr + c'$ for any constants $c < 3$ and c' . The basic idea used for their solution is to create two basic data structures within the network: a “directed ring” (that is, a firing squad in the shape of a circle with the caveat that soldiers may only listen to the man on their left and speak to the man on their right) and a “ring-of-trees” (that is, networks which include a loop, containing the root, whose length is at least as great as the maximum distance from the root to any processor).
- FSSP for general directed networks:
Each soldier is modeled by a finite automaton that has a fixed number of input terminals and a fixed number of output terminals. Edges connect output terminals to input terminals. Each output terminal is connected to at most one input terminal (that is, fan-out is at most 1). The first solution of this variation is by Kobayashi [16]. The firing time of his solution is an exponential function of the number n of soldiers, and the solution essentially uses the same structures as Nishitani and Honda. Even, Litman, and Winkler [8] constructed a solution with firing time $O(n^2)$. This solution again uses the same data structures as Nishitani and Honda, but they are constructed in a faster way thanks to their invention of network-traversing “snakes.” Ostrovsky and Wilkerson [29] improved the firing time to $O(nd)$, where d refers to the diameter of the network, which remains the best to date.

1.3.3. Variations that have no minimal-time solutions. There are only a couple of “lower bound” results in the literature. The first is an impossibility result for the specific topology in which processors may have unbounded fan-out from any given output (proved in [17]); there cannot exist any solution at all for this particular

topology. Additionally, Jiang [14, 13] showed that the FSSP for the bilateral ring with an arbitrary number of roots has no solution. In this case, all the soldiers may potentially be roots, and all roots are activated simultaneously. (Note that there are many variations of FSSP with multiple roots that have solutions. For example, Culik [5] showed that FSSP for the one-dimensional line with roots at the leftmost and rightmost ends does have a minimal-time solution.)

1.4. Our contribution. It has been a longstanding open problem to determine whether there is a network topology for which a synchronization solution is known to exist, but a *minimal-time* synchronization solution provably does not exist. After many years, the contribution of this paper is to finally answer this question in the affirmative under the assumption that $P \neq NP$. In this paper, we show that 3PATH, g-3PATH, and 3REG have no minimal-time solutions if $P \neq NP$. Hence, of the eight variations of FSSP mentioned above, at least three are highly unlikely to have minimal-time solutions.

The main result of this paper is the following theorem.

THEOREM 1.1. *If $P \neq NP$, then there does not exist a minimal-time solution to FSSP for the network topology introduced below as variation 3PATH, the direct three-dimensional analogue of Kobayashi's variation 7 from [18].*

Note that, by this theorem, the straightforward complexity-theoretic statement $P \neq NP$ implies a result about the absence of minimal-time solutions (i.e., instantiations of automata) for synchronization, two seemingly unconnected subjects. Or, equivalently, the existence of a minimal-time solution for a distributed computation synchronization primitive implies the purely complexity-theoretic result $P = NP$.

In order to prove this theorem, we need to define the three-dimensional equivalent to a problem proposed in [18, 19], the three-dimensional path extension problem (or 3PEP), which we define rigorously in section 2. The proof proceeds in steps as follows.

1. We show that a minimal-time solution for FSSP on variation 3PATH implies that there exists a deterministic Turing machine that can solve the problem 3PEP in polynomial time.
2. We prove that if $3PEP \in P$, then two simpler versions of the 3PEP problem, 1CUBE and 2CUBE, are also solvable in polynomial time. The fact that the reduction below is polynomial-time depends heavily on these two problems also being polynomial-time.
3. Given the result in step 2, we show that $3PEP \in P \rightarrow HAM \in P$, where HAM represents a restricted NP-complete version of the Hamilton path problem.

Aside from being theoretically interesting, this result is important to the field of parallel processing, as fast network processor synchronization is a fundamental computing primitive. Indeed, decades of research have focused on extending the network topologies for which there exist solutions [8, 28, 16, 29], minimizing the time and space complexity [11, 22, 38, 39], and even proving the asymptotic time equivalence of a number of other common network problems to FSSP [10].

1.5. Additional remarks. Before concluding this section we should mention some results by Mazoyer and by Schmid and Worsch and make an additional comment on the formulation of FSSP we use in this paper.

1.5.1. Results by Mazoyer and by Schmid and Worsch. In section 1.3, we surveyed known results on variations of FSSP that are generalizations of the original FSSP to more general network topologies. However, if we do not restrict ourselves to these types of variations, we know at least two others that have solutions yet no

minimal-time solutions, and the nonexistence of minimal-time solutions can be shown without a complexity-theoretical assumption such as $P \neq NP$.

Mazoyer [24] considered the following variation of FSSP. There is only one general and one soldier. The information exchange between the two nodes takes τ units of time (where τ is a positive integer). The state of a given node at time t is determined by the state of the node at time $t - 1$ and the state of the other node at time $t - \tau$. Neither node knows the value τ . The minimum firing time is a function of τ , and the notion of the minimal-time solutions is naturally defined. Mazoyer proved that this variation of FSSP has a solution yet no minimal-time solution.

Schmid and Worsch [33] considered a variation of the original FSSP on the bidirectional line such that each node may be either a general or a soldier, and generals are activated independently. A problem instance of this variation is specified by the following values: (1) the number n of nodes, (2) the number k of generals ($1 \leq k \leq n$), and (3) for each i ($1 \leq i \leq k$) the position p_i of the i th general ($1 \leq p_i \leq n$) and the time t_i at which the i th general is activated ($0 \leq t_i$). All the nodes start with the quiescent state, and for each i the node at the position p_i is forced to enter the state of a general at time t_i . They proved that this variation has a solution but has no minimal-time solution.

1.5.2. The formulation of FSSP. In this section, we give a more formal description of the FSSP. First, we review the classical formulation and then introduce and justify our slight modification.

The usual/classical formulation of FSSP can be summarized as follows. Each node of a network is a copy of a finite automaton. The set of the states of the finite automaton includes at least three different states, the *general state* G , the *quiescent state* Q , and the *firing state* F . Each input terminal of a node receives the state of the node to which the terminal is connected or a special symbol $\#$ if the input terminal is open. The state of a node at time $t + 1$ is completely determined by its state and the states of its adjacent nodes at time t . A node in the state Q remains in that state until the value of at least one of its input terminals is neither G nor $\#$. At time 0, the state of a node is G or Q depending on whether or not it happens to be the root. The goal of FSSP for a network topology is to design the finite automaton so that, for each network of that network topology, all of the nodes enter the state F simultaneously for the first time.

As a model of current network synchronization algorithms, this formulation has two problems. The first is that there is only one general state G , and hence the root cannot know its boundary condition (the existence or nonexistence of adjacent nodes for each input terminal) at time 0. In reality, the time for a node to check its boundary condition is negligibly small compared with the time for an information exchange between adjacent nodes. Hence it is natural to assume that the root can know its boundary condition at time 0. The second problem is that the general state G and the firing state F must be different, and hence the root cannot fire at time 0 even if the network has only one node. By the same reasoning, it is natural to assume that the root can fire at time 0 when the network has only one node.

Based on these considerations we modify the formulation of FSSP in the following way. First, we allow the automaton to have more than one general state. The general state to be used is uniquely determined by the boundary condition of the root. This modification solves the first problem. Second, instead of using one firing state F , we specify multiple firing states. A general state may be also specified as one of the firing states. However, the quiescent state Q cannot be specified as a firing state. All the

nodes should enter some firing state simultaneously for the first time. Different nodes may enter different firing states. If we specify the general state that corresponds to the root having no adjacent nodes as one of the firing states, the root can fire at time 0. Hence this modification solves the second problem.

We show the proofs for our main results (Theorems 1.1 and 4.3) using this modified formulation of FSSP. However, all of these results are also true for the usual formulation of FSSP. At the end of section 4 we will briefly explain how to modify the proofs for the other formulation.

The rest of the paper is organized as follows. In section 2, we rigorously define our new variation 3PATH. In section 3, we outline steps 1, 2, and 3 of the proof of Theorem 1.1. In section 4, we present the generalizing theorems for variations g-3PATH and 3REG and the corresponding proofs. Finally, section 5 explores possible avenues of further research.

2. Variation 3PATH. In this section, we will introduce the new network topology 3PATH.

We can place a processor on any point $(x_1, x_2, x_3) \in \mathcal{Z}^3$ in the three-dimensional grid. We say that two processors in positions (x_1, x_2, x_3) and (y_1, y_2, y_3) are *adjacent* if and only if $|x_i - y_i| = 1$ for exactly one value of i and for all $j \neq i$, $x_j = y_j$. Note that processors have six available ports for adjacent processors: North, South, East, West, Up, and Down.

We can abstract this type of network by a three-dimensional orthogonal grid graph in which certain vertices are marked. Each marked vertex corresponds to the position of a processor, and each edge corresponds to a bidirectional link between processor ports.

DEFINITION 2.1 (variation 3PATH). *We define a path in the infinite three-dimensional orthogonal grid \mathcal{Z}^3 as follows. A path is a sequence of distinct vertices $p_1, p_2, \dots, p_n \in \mathcal{Z}^3$ that satisfy the following properties. For any $1 \leq i < n$, p_i is adjacent to p_{i+1} . With the exception of p_1 and p_n , all vertices in the path must have exactly two adjacent vertices. If $n \geq 2$, p_1 and p_n must have exactly one adjacent vertex each. (Diagonal vertices, though allowed, are not considered adjacent by the definition above.) The root processor must be placed at p_1 . If two processors are placed in adjacent positions, we assume that the adjacent port is connected by a bidirectional wire; only adjacent processors can be connected.*

Note that this definition implies that the configuration is connected in the sense that a signal released by any processor must be able to eventually reach any other processor in the network via data transfers passed back and forth through the processors' ports.

The length of a path $p_1 p_2 \dots p_n$ is defined to be n .

This definition corresponds to the intuitive definition of a self-avoiding network path in three dimensions. The path is “self-avoiding” in the following sense: not only must the path not intersect itself, but it cannot even become adjacent to itself. If it does, then there must exist some vertex with three or more neighbors (or the boundary conditions must be violated).

A solution to this variation of the FSSP trivially exists because solutions have been found for arbitrary undirected (and directed) networks. The same solutions will function on this subvariation, just not in minimal time.

3. The results. In this section, we will outline steps 1, 2, and 3 of the proof of Theorem 1.1.

3.1. A minimal-time solution to variation 3PATH implies 3PEP ∈ P.

This section will be devoted to step 1 of the proof of Theorem 1.1. First, we need to define the problem 3PEP.

DEFINITION 3.1 (3PEP). *A problem instance will be a path as in Definition 2.1. We assume that $p_1 = (0, 0, 0)$ and represent a path as a sequence of directions: North, South, East, West, Up, and Down. The decision problem 3PEP is as follows: Is it possible to extend the given path instance to double its length from its end (and still remain a path as in Definition 2.1)?*

This is precisely the three-dimensional analogue of the two-dimensional path problem presented in [18, 19].

The proof that a minimal-time FSSP solution to variation 3PATH implies 3PEP ∈ P is a direct three-dimensional adaptation of that given in [18, 19], and we will present only a short explanation here for the sake of completeness. For a much more thorough treatment, we recommend consulting the stated references. In the following, whenever we refer to a “path,” we mean a path in the sense of Definition 2.1.

Let $p_1p_2 \dots p_n$ be a path. For $1 \leq i \leq n$, let the value $e(p_1p_2 \dots p_n, i)$ be the value of the length of the longest possible consistent extension to the path $p_1p_2 \dots p_i$ if $i < n$ (i.e., the maximum value³ m such that there exists a path $p_1p_2 \dots p_i p_{i+1}q_2 \dots q_m$) and 0 if $i = n$.

LEMMA 3.2. *Let $p_1p_2 \dots p_n$ be a path, and let $i_0 = \min\{i : 1 \leq i \leq n, i \geq e(p_1p_2 \dots p_n, i)\}$. Intuitively, i_0 represents the label of the first vertex after which point we are guaranteed that any signal released from the root has traversed at least half the total “possible” path length.*

Then a minimal-time FSSP solution for variation 3PATH, if it exists, running on the network $p_1p_2 \dots p_n$, will fire the network in number of clock steps equal to either $2i_0 - 1$ if $i_0 = e(p_1p_2 \dots p_n, i_0)$ or $2i_0 - 2$ if $i_0 > e(p_1p_2 \dots p_n, i_0)$.

Proof. First, we show that the firing time of any solution A to FSSP for variation 3PATH is at least the value stated in the lemma.

Suppose that $i_0 = e(p_1p_2 \dots p_n, i_0)$. This implies that $i_0 < n$ and that there is a path of the form $p_1p_2 \dots p_{i_0}p_{i_0+1}q_2 \dots q_{i_0}$. Consider running the solution A on both of the paths $\alpha = p_1p_2 \dots p_n$ and $\beta = p_1p_2 \dots p_{i_0}p_{i_0+1}q_2 \dots q_{i_0}$. Then, at the time $2i_0 - 2$, the states of A at p_1 in α and p_1 in β are the same, and the state of A at q_{i_0} in β is the quiescent state Q . Hence the solution A cannot fire on $\alpha = p_1p_2 \dots p_n$ at the time $2i_0 - 2$. Therefore the firing time of A for $\alpha = p_1p_2 \dots p_n$ is at least $2i_0 - 1$.

Now suppose that $i_0 > e(p_1p_2 \dots p_n, i_0)$. If $i_0 = 1$, the firing time is at least $2i_0 - 2$ because $2i_0 - 2 = 0$. Suppose that $i_0 \geq 2$. Then we have $i_0 \leq e(p_1p_2 \dots p_n, i_0 - 1)$. Hence there exists a path of the form $\beta = p_1p_2 \dots p_{i_0-1}p_{i_0}q_2 \dots q_{i_0}$. As in the previous case, using this we can show that the firing time of A for $\alpha = p_1p_2 \dots p_n$ is at least $2i_0 - 2$.

Next we show that there exists a solution whose firing time for $p_1p_2 \dots p_n$ is at most the value stated in the lemma. We consider only the case where $i_0 < n$. The modification for the case where $i_0 = n$ is easy. Assume we are given some solution A to FSSP for variation 3PATH that is not necessarily minimal-time for all paths. For any path $p_1p_2 \dots p_n$, we can modify the solution to a new solution A' which is minimal-time for that particular path. Along with all other signals sent by A , send a searcher signal down the path until it reaches processor p_{i_0} .

³If a maximum value m does not exist (i.e., if the path could extend indefinitely), we define $e(p_1p_2 \dots p_n, i) = \infty$.

Suppose that the searcher signal⁴ finds that the current problem instance has the same initial sequence as the special path $p_1p_2 \dots p_{i_0}p_{i_0+1}$. The searcher signal finds this at time $i_0 - 1$ arriving at p_{i_0} .⁵ Then the distance between p_{i_0} and any vertex p_i behind it ($1 \leq i < i_0$) is at most $i_0 - 1$, and the distance between p_{i_0} and any possible vertex ahead of it is at most i_0 if $i_0 = e(p_1p_2 \dots p_n, i_0)$ and at most $i_0 - 1$ if $i_0 > e(p_1p_2 \dots p_n, i_0)$. Hence, as soon as the searcher signal arrives at p_{i_0} at time $i_0 - 1$, it sends a signal back to both ends of the line saying “fire after i_0 more steps” if $i_0 = e(p_1p_2 \dots p_n, i_0)$ and “fire after $i_0 - 1$ more steps” if $i_0 > e(p_1p_2 \dots p_n, i_0)$. After each time step, the signal decreases its countdown by one. When the countdown reaches 0, every processor simultaneously fires. The firing time is $(i_0 - 1) + i_0 = 2i_0 - 1$ if $i_0 = e(p_1p_2 \dots p_n, i_0)$ and $(i_0 - 1) + (i_0 - 1) = 2i_0 - 2$ if $i_0 > e(p_1p_2 \dots p_n, i_0)$.

If the searcher signal fails to find that the current problem instance has the same initial sequence, the algorithm simply uses the signals from the original protocol, A .

Thus, the minimum firing time of the network given by the path $p_1p_2 \dots p_n$ is at most $2i_0 - 1$ if $i_0 = e(p_1p_2 \dots p_n, i_0)$ and at most $2i_0 - 2$ if $i_0 > e(p_1p_2 \dots p_n, i_0)$. \square

THEOREM 3.3 (step 1 in the proof of Theorem 1.1). *If a minimal-time FSSP solution to variation 3PATH exists, then 3PEP \in P.*

Proof. Assume that a minimal-time FSSP solution MIN to variation 3PATH exists. Let $p_1p_2 \dots p_n$ be an instance of 3PEP, encoded appropriately.

Consider the path $p_1p_2 \dots p_n$. At the vertex p_n , there are five different directions for extending the path. If it is impossible to extend the path by one more vertex from p_n , then clearly the answer to 3PEP is NO.

We will assume that the path can be extended in some direction to $p_1p_2 \dots p_nr$. Arrange $n + 1$ copies of the solution MIN on the $n + 1$ vertices of $p_1p_2 \dots p_nr$. Simulate the algorithm and note the firing time. Let $i_0(r)$ be the value $\min\{i : 1 \leq i \leq n + 1, i \geq e(p_1p_2 \dots p_nr, i)\}$. Lemma 3.2 yields the following results:

- If the firing time is greater than or equal to $2n$, then either $2i_0(r) - 1 \geq 2n$ or $2i_0(r) - 2 \geq 2n$, and hence $i_0(r) - 1 \geq n$. This together with $i_0(r) \leq n + 1$ implies $i_0(r) = n + 1$. From $i_0(r) \leq e(p_1p_2 \dots p_nr, i_0(r) - 1)$, or equivalently $n + 1 \leq e(p_1p_2 \dots p_nr, n)$, we know that $p_1p_2 \dots p_nr$ has an extension of length at least $n + (n + 1) = 2n + 1$. Therefore the answer to 3PEP is YES.
- If the firing time is $2n - 1$, then $n = i_0(r) = e(p_1p_2 \dots p_nr, n)$. Hence $p_1p_2 \dots p_nr$ has an extension of length $n + n = 2n$. Therefore the answer to 3PEP is YES.
- If the firing time is $2n - 2$, then $n = i_0(r) > e(p_1p_2 \dots p_nr, n)$. Hence the length of any extension of $p_1p_2 \dots p_nr$ is at most $n + (n - 1) = 2n - 1$. Therefore the path $p_1p_2 \dots p_nr$ cannot be extended to an extension of length $2n$, and we need to check other possible directions for r .
- Finally, if the firing time is less than or equal to $2n - 3$, then either $2i_0(r) - 1 \leq 2n - 3$ or $2i_0(r) - 2 \leq 2n - 3$, and hence $i_0(r) \leq n - 1$. From $i_0(r) \geq e(p_1p_2 \dots p_nr, i_0(r))$ we know that the length of any extension of $p_1p_2 \dots p_{i_0(r)}p_{i_0(r)+1}$ is at most $i_0(r) + i_0(r) \leq 2n - 2$. Therefore the path

⁴The size of the searcher signal and the other signals sent by this algorithm depend critically on the number i_0 . If we attempt to create a minimal-time solution for *all paths* using this construction, we will run into a serious problem. As the paths grow in size, the size of this number i_0 may become unbounded. However, if we fix some particular i_0 and tailor a specific solution for that i_0 , even though the size of the automata will depend on i_0 , once we fix its value, it is a constant. Therefore, for any *fixed* path, there exists a solution that is minimal-time for that particular path.

⁵Note that the signal need not reach p_{i_0+1} in order to verify its existence. Once it reaches processor p_{i_0} , the processor can “inform” the signal if it is connected in the correct direction.

$p_1p_2 \dots p_n$ cannot be extended to twice its length, and hence the answer to 3PEP is NO.

So to determine the answer to 3PEP for the path $p_1p_2 \dots p_n$, we can have the deterministic Turing machine simulate the operation of a network on $p_1p_2 \dots p_nr$ for each r , using the minimal-time FSSP solution MIN . We check every possible direction for r (only five at most). If a single choice for r yields a definitive answer, then we return that answer as the answer to 3PEP. Otherwise, the answer is NO. This clearly takes polynomial time in the input size. \square

3.2. 3PEP \in P \rightarrow 1CUBE \in P and 3PEP \in P \rightarrow 2CUBE \in P. In this section, we complete step 2 of Theorem 1.1. The final step in the proof of Theorem 1.1 is a reduction that depends heavily on these two problems being polynomial-time solvable.

We will make use of four basic building blocks illustrated in Figures 3.1, 3.2, 3.3, and 3.4. They are closed walls, open walls, spigots, and connectors. We will show how they are used to make more complicated structures later in this section. We make the convention in all our constructions below that all walls are square as they are above. (This is not essential to the proof, but it makes explanations conveniently shorter.) A path will never pass through a closed wall and can only pass through an open wall through the 6×4 opening. Note that the basic building blocks are all composed of a single path.

We now describe the formation of the “2-holed cube.” A 2-holed cube will consist of 4 closed walls, 2 open walls, 2 spigots, and 7 paths connecting the various building blocks together. This structure is confusing to view in a single picture all at once. We will describe the steps used to create it using a sequence of pictures. Figures 3.2 and 3.3 illustrate how the spigot should be placed so that the multiple potential openings in the open wall will not “leak.” Figure 3.5 illustrates the appropriate placement of the cube walls.

We will now define the decision problems 1CUBE and 2CUBE.

DEFINITION 3.4 (1CUBE). *A problem instance will be a number n represented in unary⁶ and a positive integer K . The decision problem 1CUBE is as follows: Does an $n \times n \times n$ 2-holed cube in \mathcal{Z}^3 admit an avoiding⁷ path (as in Definition 2.1) of length at least K that begins in the middle of the spigots of one of the open walls, enters the cube, and remains inside the cube? The path may end at the center of the spigot of the other open wall but may not pass through.*

DEFINITION 3.5 (2CUBE). *A problem instance will be a number n represented in unary and a positive integer K . The decision problem 2CUBE is as follows: Does an $n \times n \times n$ 2-holed cube in \mathcal{Z}^3 admit an avoiding path (as in Definition 2.1) of length at least K that begins at the center of the spigot of one of the open walls, enters the cube, and ends at the center of the spigot of the other open wall?*

THEOREM 3.6. 3PEP \in P \rightarrow 1CUBE \in P.

Proof. Assume that 3PEP is solvable in polynomial time. Consider the problem 1CUBE with parameters n and K . In the three-dimensional grid, we form a 2-holed cube of size $n \times n \times n$. We form this cube from a single path beginning outside the cube and ending outside the cube in the position adjacent to the center of one of the

⁶The reason we represent the number n in unary here and in Definition 3.5 is that we wish to be able to construct the cube in \mathcal{Z}^3 as a polynomial-time part of the problem instance. Note that in Definition 3.1, the path is also explicitly constructed.

⁷The term *avoiding* refers to the fact that the path must never become adjacent to itself or the walls of the cube. We use this term in the same manner in Definition 3.5.

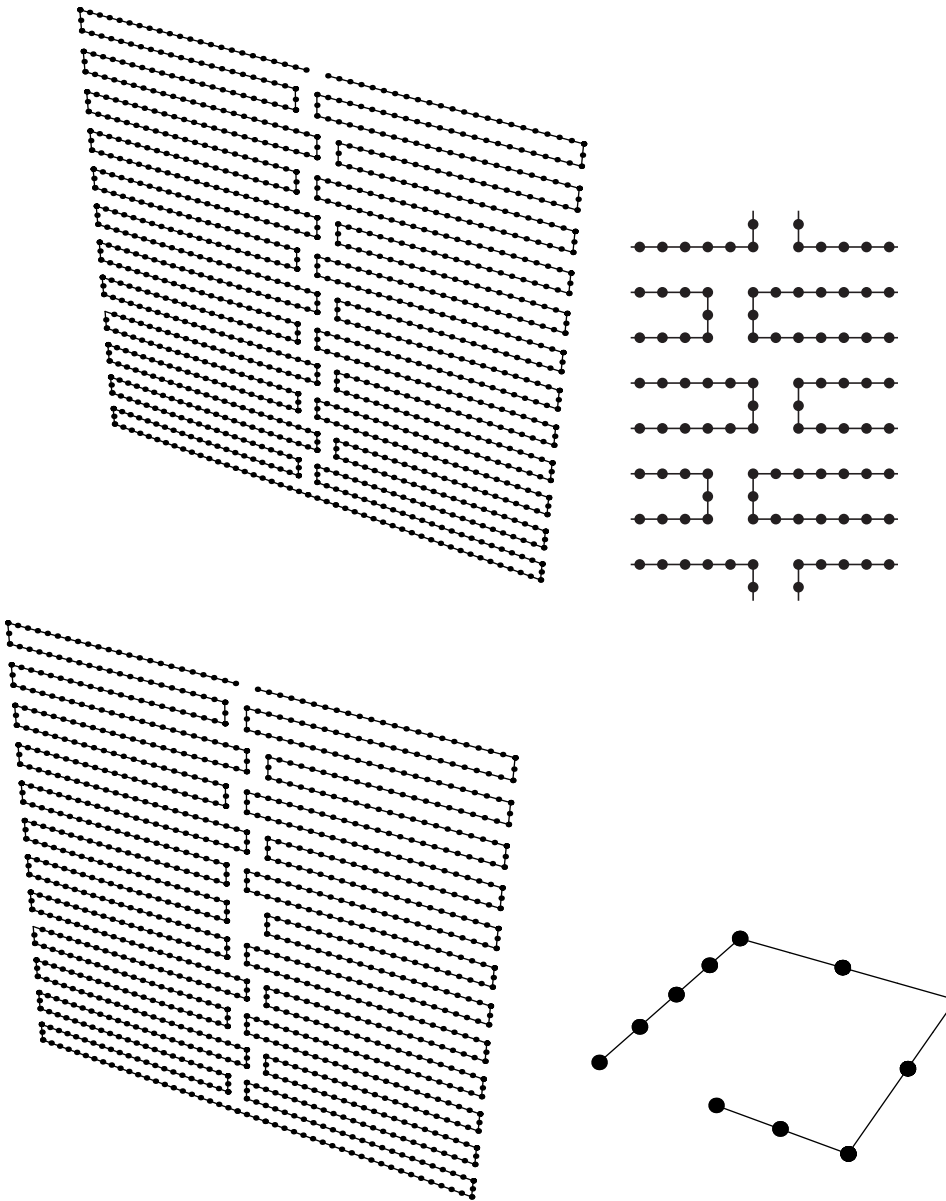


FIG. 3.1. This figure illustrates some of the simple building blocks we will use to create the more complicated structures below. The upper left picture is a “closed” wall formed by a single path. Just to the right, we provide a snapshot of the center of the closed wall. Note that even though there is not a solid mass of vertices, a continuation of the path could not possibly pass through the wall, as it would violate the definition of a path (Definition 2.1).

Below these two is an “open” wall and a “spigot.” Note that because the height of the hole in the center is 6 edges tall and 4 edges wide, it is possible for the path to pass through the center of this structure without violating any path properties. (We provide a close-up of the center of the open wall in Figure 3.2.) Unfortunately, there are several potential places for a path to pass through our open wall. We would prefer that there be only a single unique entrance/exit point for any open wall; we therefore place a “spigot” at the opening of every open wall. Of course, it must be placed 2 units from the wall opening so as not to violate the path definition. This concept will be further outlined in Figure 3.2.

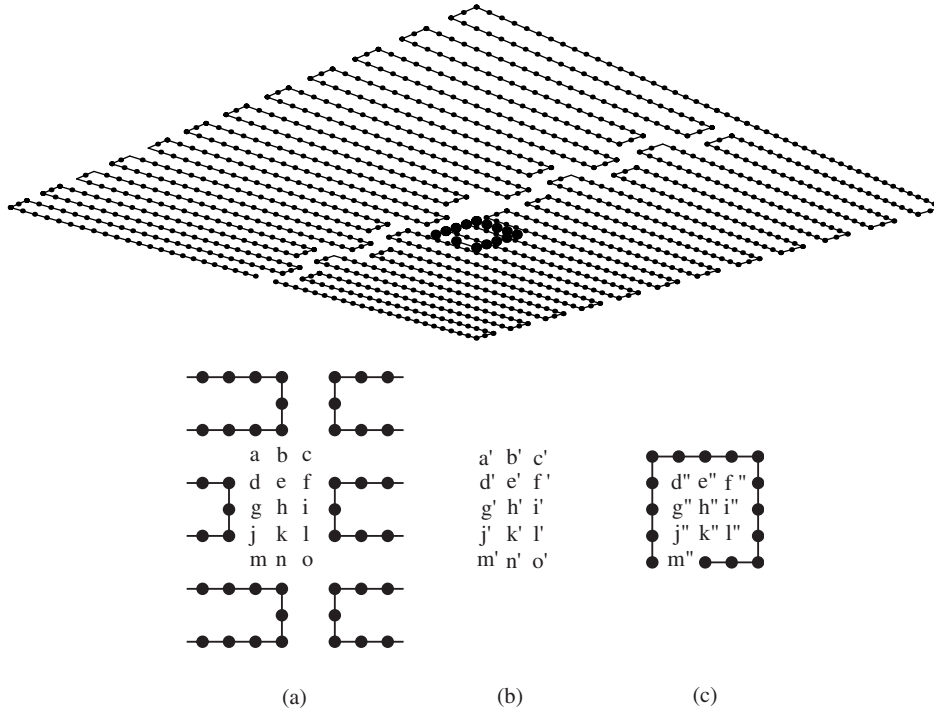


FIG. 3.2. This figure shows the shapes of holes and spigots and their relative placement. Above is a figure illustrating the general idea of where the spigot would be placed: 2 units away from the wall in the center of the hole. The lower figure makes this more precise:

Three figures (a), (b), (c) represent three adjacent planes. A snapshot of the center of an open wall is on the plane (a) and a spigot is on the plane (c). The plane (b) is between the two planes (a) and (c) and is empty. Three positions d, d', d'' are on one straight line, and similarly for the other letters. We may regard the hole in (a) as a rectangle with dimensions 6 edges tall and 4 edges wide containing 15 inner grid points a, b, c, \dots, o . For a path to enter into the wall through the hole, the path must pass through the center h'' of the spigot. After that position, there are numerous ways for the path to proceed, e.g., $h'' \rightarrow h' \rightarrow h \rightarrow \text{inside}$, $h'' \rightarrow h' \rightarrow h \rightarrow e \rightarrow \text{inside}$, $h'' \rightarrow h' \rightarrow h \rightarrow k \rightarrow \text{inside}$, $h'' \rightarrow h' \rightarrow e' \rightarrow e \rightarrow \text{inside}$, and $h'' \rightarrow h' \rightarrow k' \rightarrow k \rightarrow \text{inside}$. Note that if we do not place the spigot, a path can pass through the hole at least four times, for example, enter through c , exit through e , enter through k , and exit through o . The spigot forces the path to enter (or exit) only once through one fixed position h'' .

spigots. We create an extension from the beginning of the path in such a way that the path begins at a distance 2 away from the center of the other spigot. See Figure 3.6 for an illustration of the beginning of the path “plugging” one of the opening spigots. Note that the path, when extended from the end, must lead into the cube. Either the parameter K is less than the number of processors in the current path or it is not. If K happens to be less than the current number of processors in the path, then for large enough n , the answer to 1CUBE is YES because the volume inside the cube is $\Theta(n^3)$ and $K = O(n^2)$.

Now assume that K is greater than or equal to the number of processors in the current path. Extend the beginning of the path outwards until the total path length is precisely equal to K . We can now use the polynomial-time oracle for 3PEP to determine the answer for the 1CUBE problem. If we run the 3PEP algorithm on the constructed path, the answer will be the same as that for 1CUBE. \square

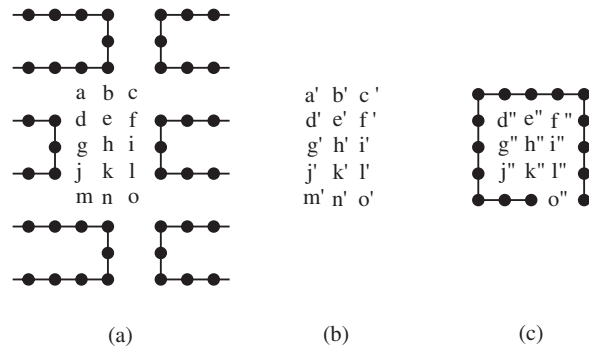


FIG. 3.3. This figure illustrates a potential pitfall of the spigot placement: If we place them in the way illustrated here rather than correctly (as in Figure 3.2), a path can enter into the wall through the positions o' and o without passing through the spigot. For one fixed orientation of the hole there are eight ways to place the spigot, and we must avoid two of them, those with the vacant position of the spigot at c'' and o'' .

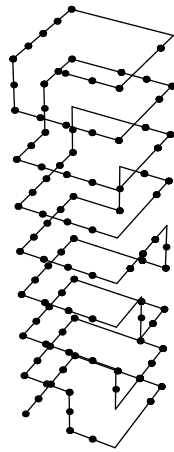


FIG. 3.4. This figure illustrates a “connector” that connect two open walls together. Note that the ends of the connector are simply two spigots.

Note that because the 1CUBE decision problem is solvable in polynomial time, it is trivial to construct a polynomial-time algorithm to determine the length of the longest path that enters an $n \times n \times n$ cube once through the middle of a face and remains inside.

THEOREM 3.7. $3PEP \in P \rightarrow 2CUBE \in P$.

Proof. Assume that 3PEP is solvable in polynomial time. Consider the problem 2CUBE with parameters n and K . In the three-dimensional grid, we form a 2-holed cube of size $n \times n \times n$. This cube is again formed from a single path beginning outside the cube and ending outside the cube in the position adjacent to the center of one of the spigots; thus the path, when extended, again must lead into the cube. We also form another $n^2 \times n^2 \times n^2$ 2-holed cube at a distance n away such that the opening for the $n \times n \times n$ cube lines up exactly with the uncovered opening from the larger cube. We then connect the two with a connector as illustrated in Figure 3.7. Using the same construction as in the proof of Theorem 3.6, we close off the other opening

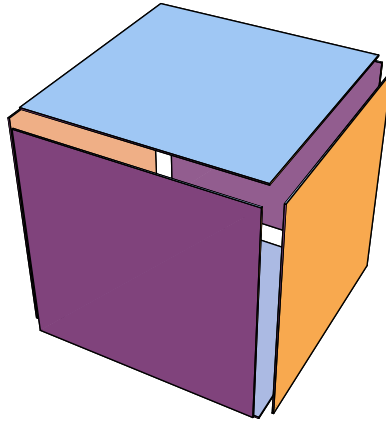


FIG. 3.5. This figure illustrates the placement of the walls for the formation of the cube structure. We choose four of the walls to be closed and the other two open. The two open walls must be adjacent. If (x_1, y_1, z_1) , (x_2, y_2, z_2) , (x_3, y_3, z_3) are the coordinates of the three vertices at the corner of the cube, then we specify that for each i, j ($1 \leq i < j \leq 3$), of the three values $|x_i - x_j|$, $|y_i - y_j|$, $|z_i - z_j|$, two have value 1 and the other has value 0. Hence, a path can pass through neither the corner nor the edge of a cube. Each opening in the open walls is a rectangle of size 4×6 . We require that the two open walls are always placed so that the four sides of length 6 of the rectangles are always parallel.

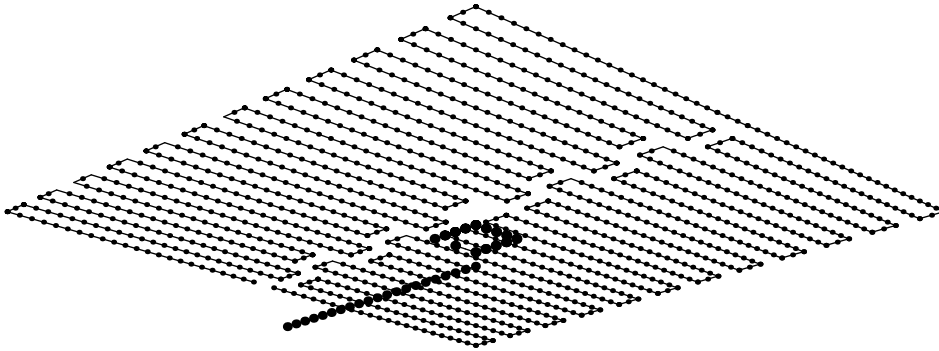


FIG. 3.6. This figure illustrates a spigot plug. Note that the path begins at a distance 2 from the center of the spigot. Because the only exit from the spigot is through the center, this effectively stops any path inside the cube from exiting through the hole.

of the larger cube. The entire structure is built from a single path.

Given the result from Theorem 3.6 and the subsequent remark, it is possible to determine the length of the longest path that can be contained within the one-holed cube of size $n^2 \times n^2 \times n^2$. Call this length L . We also know that the distance from the exit of the $n \times n \times n$ 2-holed cube to the entrance of the other is n .

Note that for large enough n , the current path length must be strictly less than the quantity $L + n + K$ because $L = \Theta(n^6)$. We can therefore extend the beginning of the current path until it has length exactly $L + n + K$. We can now use the polynomial-time oracle for 3PEP to determine the answer for the 2CUBE problem.

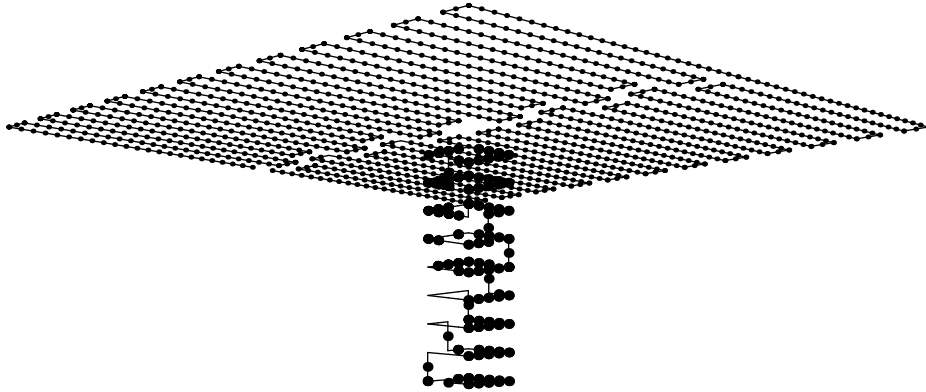


FIG. 3.7. This figure illustrates the connection of open cube faces via a connector, one of the basic building blocks. Note that the connector stops at a distance of 2 units away from the faces of the cube. The connector, by construction, conveniently has a spigot on both ends. Note that we must be very careful with the placement of the spigots on the holes in the cubes as illustrated by Figures 3.2 and 3.3. A misplacement may allow a path to enter a cube via a “bad” path. (We need the entrances and exits to be exactly the same for each cube so that the problems 1CUBE and 2CUBE are well defined.) Luckily our construction allows us to move cubes and connectors by a constant amount, so we can always assume that the spigots connect in exactly the right way.

If we run the 3PEP algorithm on the constructed path, the answer will be the same as that for 2CUBE. \square

Note that because the 2CUBE decision problem is solvable in polynomial time, we can also trivially construct an algorithm for determining the length of the longest path completely enclosed by the $n \times n \times n$ cube that is required to begin at the “entrance” and end at the “exit.”

3.3. 3PEP \in P \rightarrow HAM \in P. In this section, we will justify step 3 in the proof of Theorem 1.1.

In order to do so, we need to introduce an operation on graphs in \mathcal{Z}^3 , called a *blow-up*, by a factor of k . The idea behind this operation is simple. We apply the linear map $(x_1, x_2, x_3) \mapsto (kx_1, kx_2, kx_3)$ to every point in \mathcal{Z}^3 . Note that the blow-up operation preserves slopes of lines but increases distances by a factor of k .

THEOREM 3.8 (step 3 in the proof of Theorem 1.1). *Let HAM be the following special case of the Hamilton path problem. As stated in [9], the Hamilton path problem [GT39] remains NP-complete under the assumption that the graph G is planar, cubic, 3-connected, and has no face with fewer than five edges. (A cubic graph is one with every vertex of degree 3. We will not need the latter two properties.) Then we have the following result: 3PEP \in P \rightarrow HAM \in P.*

Proof. Assume that we are given a planar, cubic graph G and that 3PEP \in P. We first choose two vertices in G , v_{start} and v_{end} , and remove two edges from each vertex, leaving only one edge remaining per vertex. These vertices will correspond to the start and end of a potential Hamilton path in the graph G . Call the new graph G' .

Our first nontrivial goal will be to illustrate how it is possible to build a three-dimensional connected structure out of a single path that has the same connectivity properties as G' . The structure will contain an “inside” and an “outside” such that a path originating on the inside of the structure cannot possibly make it to the outside and vice versa. The beginning of the path (root) will be required to be outside the

structure, and the end of the path will be inside the structure. We will then show how it is possible to solve the Hamilton path problem on G' utilizing a solution to the 3PEP problem.

An integer-grid straight-line drawing of a planar graph G is an embedding of G in the two-dimensional integer grid such that the vertices of G are grid points and the edges of G are noncrossing straight lines. In [7, 6], it is shown how an n -vertex planar graph can be embedded as an integer grid straight-line drawing into the plane in $O(n^2)$ space, requiring a grid of size $O(n) \times O(n)$, and polynomial time. In fact, [4] does the same in linear time. Many other references on graph embedding can be found in [2]. Embed the graph G' in the xy -plane (i.e., $z = 0$) in \mathcal{Z}^3 in this way. (Note that it is possible to create the structure outlined below without making use of the planarity and the integer-grid straight-line drawing. Eliminating these restrictions will not change the ultimate result, though it will increase the length of the explanation significantly by forcing us to explicitly describe a separate method for the relevant constructions.)

In the following discussion, we will assume that n is large enough to dwarf all constants hidden by the asymptotic notation.

We now have a planar graph in \mathcal{Z}^3 with no crossing edges. Blow-up \mathcal{Z}^3 by a factor of n^5 . Note that, by elementary geometry, any vertex or line without an endpoint at v must have been at least distance $\Omega(\frac{1}{n})$ from v before the blow-up. Blow-ups increase distances by factors. Thus, for any vertex v , we can be guaranteed that all other vertices and lines without an endpoint at v are at least a distance $\Omega(n^4)$ from v . Around each vertex v , we can draw the following figure using our building blocks: (a) an “inner” $\Theta(n^3)$ -sized 2-holed cube with v at the center (following the same construction as in Theorems 3.6 and 3.7), (b) for all vertices other than v_{start} and v_{end} , a “surrounding” 3-holed cube with v at the center and with edge size $\Theta(1)$ larger than the 2-holed cube, and (c) for v_{start} and v_{end} , a “surrounding” 1-holed cube with the appropriate center vertex and size. We specify that any 3-holed cube must have each open face adjacent to the other two open faces. One of the open faces must be facing in the positive z -direction. The 1-holed cube must also have its single opening face in the positive z -direction. Both 1-holed and 3-holed cubes are required to have spigots on all open faces in the appropriate places to plug leaks. We are guaranteed that no surrounding cube comes within a distance of $\Omega(n^4)$ of any other surrounding cube. Once the cubes have been drawn, delete the vertex v . (In other words, we are replacing vertices with cubes.)

At this point, we have a three-dimensional situation in which the vertices are represented by two large cubes, one inside the other, and are connected by straight-line segments. We now concentrate on replacing the edges of the original straight-line drawing with three-dimensional structures. Our goal is to replace the straight lines that would otherwise enter the vertices with three-dimensional connectors. Of course, these straight line segments do not necessarily proceed along a rectilinear path, as the three-dimensional connectors do, and we are therefore required to use the third dimension: If a connector needs to make a rectilinear turn, we simply direct it into a small $\Theta(1)$ -sized box⁸ and open a second face of the box in the direction that we wish to turn, continuing a second connector out of this new opening. We replace every straight-line segment by an appropriate string of connectors that turn as necessary.

⁸*Terminology:* Note that a “box” is used for a turn and is always constant-sized. “Cubes” are the structures that replaced the vertices above, and all have edge size $\Theta(n^3)$. We use two different words to distinguish between these two, even though both shapes are the same; their size and utility are different.

In order to make the new three-dimensional structure have the same connectivity properties as G' , we need to guarantee that all connectors are able to enter and leave the appropriate cubes without crossing or interfering with the rest of the structure. Recall that the original graph G' was embedded in the xy -plane in \mathcal{Z}^3 . Number each of the $O(n^2)$ edges of G' . (In fact, because G' is cubic, an even better bound of $O(n)$ edges applies, but throughout this paper we will need only the $O(n^2)$ bound. The cubic nature of the graph G' is used in a different essential way in the construction: it is trivial to insert the constant-sized holes in the sides of the cubes if there only need to be three of them.) Note that if we proceed upwards in the z -direction, after $\Theta(n^3)$ distance (the maximum distance that any cube stretches outwards), the space is structure-free. Assign each edge a unique space in which to maneuver from cube to cube. Let the edge size of the “turn” box for the connectors be $c_{turn} = \Theta(1)$. Edge i will receive the grid space between $(c_{turn} + 4) * i + \Theta(n^3)$ and $(c_{turn} + 4) * (i + 1) + \Theta(n^3)$ in the z -direction.⁹ (In other words, edge i receives all grid points of the form (x, y, z) where $(c_{turn} + 4) * i + \Theta(n^3) \leq z < (c_{turn} + 4) * (i + 1) + \Theta(n^3)$.) We connect Cubes A and B with connectors as follows: upon leaving Cube A , we simply make a turn directly upwards (if necessary), move to the appropriate maneuver space, proceed to the spot above Cube B using only at most two more turns, turn downwards towards Cube B , and then turn towards Cube B (if necessary) to enter. In this way, the new three-dimensional graph has the same connectivity properties as the original graph G' .

At this point, we have a three-dimensional structure that is broken into many separate pieces. We connect these pieces into one long path G'_{3D} by connecting the beginning of one structure with the end of the next structure.¹⁰ Let the path G'_{3D} end between the inner and outer cubes of the vertex v_{start} ; thus, the path must extend into the unique connector of v_{start} with or without entering the inner cube. G'_{3D} will begin outside the enclosed structure. We now have a connected three-dimensional structure G'_{3D} that represents the graph G' in three dimensions, where the vertices are pairs of large concentric cubes, edges are three-dimensional connectors, and the entire structure is drawn with a single path originating outside the structure and terminating in such a way that any extension must continue inside the structure.

Before continuing with the analysis, we must guarantee that the number of vertices in the path represented by G'_{3D} is bounded above by a polynomial in n (the input size of G) and that the time it took to construct G'_{3D} is also polynomial in n . We omit a lengthy but fairly easy proof of both of these statements.

We now examine the structure G'_{3D} . Note that the size of each cube in the structure is $\Theta(n^3)$. Because $3PEP \in P$ by assumption, we know from Theorems 3.6 and 3.7 that $1CUBE \in P$ and $2CUBE \in P$. By the subsequent remarks, we can determine the maximum path length that a given cube of either type admits in polynomial time in n . Let this maximum path length be $L_1(n)$ for $1CUBE$ and $L_2(n)$ for $2CUBE$. Both quantities are clearly $\Theta(n^9)$. (Again, we omit the fairly trivial proof of the previous statements.)

The length of the path given by G'_{3D} is $O(n^8)$. Because the path originates outside the structure G'_{3D} we have formed, we can extend the beginning (the root) as far as we like. Extend the beginning of the path so that the length of the path

⁹The extra 4 in the expressions is a guarantee that we have a two unit buffer on either side of the connector, so the path remains true to Definition 2.1. We require the extra $\Theta(n^3)$ to make sure that we make it over the tops of all of the cubes we have already made and into the structure-free space.

¹⁰This is actually a nontrivial exercise in practice. Theoretically, however, it is trivial that it is possible to make these connections given the huge amount of space between each structure.

structure G'_{3D} ends up being $(n - 1)L_2(n) + L_1(n) - \lfloor \frac{L_2(n)}{2} \rfloor$. Call this new path $G'_{3D,ext}$. We then ask the following question (3PEP): Can we extend the path given by $G'_{3D,ext}$ to double its length? (In other words, can we extend $G'_{3D,ext}$ by length at least $(n - 1)L_2(n) + L_1(n) - \lfloor \frac{L_2(n)}{2} \rfloor$ and still have the resulting structure satisfy the properties for a path in Definition 2.1?)

We claim that a Hamilton path exists in the graph G' if and only if the answer to this question for the corresponding path $G'_{3D,ext}$ is YES.

There are two cases to consider.

1. Assume that there is a Hamilton path in the graph G that starts at v_{start} , uses the edge from v_{start} that was not removed, and terminates at v_{end} via the single edge not removed. Then we can guarantee that there exists a path extension of $G'_{3D,ext}$ that enters and exits exactly $n - 1$ inner cubes and enters and possibly remains inside one additional inner cube. The length of this path extension can be made to exceed $(n - 1)L_2(n) + L_1(n)$. Therefore the maximal path extension of $G'_{3D,ext}$ must have at least this length, clearly greater than $(n - 1)L_2(n) + L_1(n) - \lfloor \frac{L_2(n)}{2} \rfloor$.
2. Assume that there is *not* a Hamilton path in the graph G that starts at v_{start} , uses the edge from v_{start} that was not removed, and terminates at v_{end} via the single edge not removed.

We claim that any path extension of $G'_{3D,ext}$ can pass through at most $n - 1$ outer cubes. To see this, assume that $G'_{3D,ext}$ has a path extension that passes through all of the n outer cubes. If the extension ends in a connector, delete its tail in the connector so that the extension ends in an outer cube. Then this extension naturally determines a path in G' such that (i) it starts with v_{start} , (ii) it touches all nodes of G' , and (iii) it never passes through an edge more than once. This path starts at v_{start} and proceeds through the only edge from v_{start} and eventually through the only edge to v_{end} , finally terminating at v_{end} . Each vertex other than v_{start} and v_{end} has three edges; hence the path passes through each exactly once. However, this means that the path is a Hamilton path of G' , and this is a contradiction.

Thus, a maximal path extension of $G'_{3D,ext}$ can only enter at most $n - 1$ outer cubes and hence only at most $n - 1$ inner cubes. Consider an extension to $G'_{3D,ext}$ that does not enter every inner cube in the drawing. The total length of all of the “connector” edges is at most $O(n^8)$ because there are at most $O(n^2)$ edges, each of which has “length” at most $O(n^6)$. If we skip at least one cube, then the most the path can be is either $(n - 2)L_2(n) + L_1(n) + O(n^8)$ or $(n - 1)L_2(n) + O(n^8)$, and both of them are at most $(n - 2)L_2(n) + L_1(n) + O(n^8) < (n - 1)L_2(n) + L_1(n) - \lfloor \frac{L_2(n)}{2} \rfloor$ because $L_2(n) \leq L_1(n)$.

In this way, we can decide whether or not the graph G' has a Hamilton path in polynomial time. To decide whether or not the original graph G has a Hamilton path, we need to perform the above decision for each combination of v_{start} , v_{end} , and edge removal. The answer to HAM for G is YES if and only if the answer is YES for at least one combination. The number of combinations is a polynomial in n . Hence, we can decide HAM for G in polynomial time under the assumption that 3PEP \in P. \square

4. FSSP for generalized 3PATH and 3REG. In this section we prove results similar to Theorem 1.1 for two other variations g-3PATH and 3REG. First, we give the definitions of these variations.

DEFINITION 4.1 (variation g-3PATH). *The FSSP for generalized three-dimensional paths in the three-dimensional grid, or g-3PATH for short, is the FSSP such*

that a problem instance is a path $p_1p_2\dots p_n$ in \mathcal{Z}^3 and the position of the root may be any vertex p_i of the path ($1 \leq i \leq n$).

DEFINITION 4.2 (variation 3REG). *The FSSP for regions in the three-dimensional grid, or 3REG for short, is the FSSP such that a problem instance is a nonempty finite connected subset X of \mathcal{Z}^3 and the position of the root may be any vertex in X . (A subset X of \mathcal{Z}^3 is said to be connected if for any p, p' in X there exists a path $p_1p_2\dots p_n$ such that $p_1 = p$, $p_n = p'$, and $\{p_1, p_2, \dots, p_n\} \subseteq X$.)*

In this section we prove the following theorem for g-3PATH and 3REG that corresponds to Theorem 1.1 for 3PATH.

THEOREM 4.3. *If $P \neq NP$, then there do not exist minimal-time solutions of g-3PATH or 3REG.*

We prove this theorem by outlining a result for g-3PATH and 3REG that corresponds to Lemma 3.2. The rest follows trivially from this observation.

The following lemma corresponds to Lemma 3.2.

LEMMA 4.4. *Let $p_1p_2\dots p_n$ be a path. We regard this path as a problem instance of g-3PATH or 3REG such that p_1 is the position of the root. Let i_0 be $\min\{i : 1 \leq i \leq n, i \geq e(p_1p_2\dots p_n, i)\}$.*

Then a minimal-time FSSP solution of variation g-3PATH or 3REG, if it exists, running on $p_1p_2\dots p_n$ will fire at time either $2i_0 - 1$ if $i_0 = e(p_1p_2\dots p_n, i_0)$ or $2i_0 - 2$ if $i_0 > e(p_1p_2\dots p_n, i_0)$.

In other words, a minimal-time FSSP solution of variation g-3PATH or 3REG runs in exactly the same time as a minimal-time FSSP solution of variation 3PATH when running on a 3PATH problem instance.

Proof. By Lemma 3.2 we know that the minimum firing time of $p_1p_2\dots p_n$ for 3PATH is $2i_0 - 1$ if $i_0 = e(p_1p_2\dots p_n, i_0)$ and $2i_0 - 2$ if $i_0 > e(p_1p_2\dots p_n, i_0)$. Moreover, g-3PATH is a subproblem of 3REG, and 3PATH is a subproblem of g-3PATH. Consequently, the minimum firing time of 3REG is at least that of g-3PATH, and the minimum firing time of g-3PATH is at least that of 3PATH. Hence, to prove the lemma it suffices to show that there is a solution of 3REG whose firing time for $p_1p_2\dots p_n$ is at most this quantity.

We can use the same searcher signal idea as in Lemma 3.2. Suppose that the searcher signal finds that the current problem instance has the same initial sequence as the special path $p_1p_2\dots p_{i_0}p_{i_0+1}$. The searcher signal finds this at time $i_0 - 1$ arriving at p_{i_0} . Then the distance between p_{i_0} and any vertex p_i behind it ($1 \leq i < i_0$) is at most $i_0 - 1$; the distance between p_{i_0} and any possible vertex ahead of it is at most i_0 if $i_0 = e(p_1p_2\dots p_n, i_0)$ and at most $i_0 - 1$ if $i_0 > e(p_1p_2\dots p_n, i_0)$ independent of whether the remainder of the configuration is a path or a connected vertex set. Thus, the remainder of the proof concludes exactly as in Lemma 3.2. \square

By Lemma 4.4, all of the previous results that were applicable to the variation 3PATH continue to apply to the variations g-3PATH and 3REG because only 3PATH configurations are used in all of the proofs.

As was stated in section 1.5.2, our main results Theorems 1.1 and 4.3 were proved using a modified formulation of FSSP. We will briefly explain how to modify the proofs for the usual formulation of FSSP.

The proof of Theorem 1.1 (the result for 3PATH) needs no modifications. The proofs of Theorem 4.3 (the results for g-3PATH and 3REG) need the following modifications.

In the proof of Lemma 4.4 we use the searcher signal. For g-3PATH and 3REG, the signal must check that the root p_1 has only one neighbor p_2 . In the usual formulation of FSSP, this needs one unit time because the root cannot know its boundary condition

at time 0, and the signal starts at p_1 at time 1 instead of at time 0. Hence, all we know on the minimum firing time is that it is either j_0 or $j_0 + 1$, where j_0 is $2i_0 - 1$ or $2i_0 - 2$ depending on whether $i_0 = e(p_1 p_2 \dots p_n, i_0)$ or $i_0 > e(p_1 p_2 \dots p_n, i_0)$.

This uncertainty of 1 on the minimum firing time influences Theorems 3.3, 3.6, and 3.7. As for Theorem 3.3, we can construct a polynomial-time algorithm for 3PEP that may give an incorrect answer when the length m of the longest extension of the given input path $p_1 p_2 \dots p_n$ satisfies $|m - 2n| \leq c_1$ for some constant c_1 . As for Theorems 3.6 and 3.7, we can construct polynomial-time algorithms that calculate values $L'_1(n)$, $L'_2(n)$ such that $|L_1(n) - L'_1(n)| \leq c_2$, $|L_2(n) - L'_2(n)| \leq c_3$ for some constants c_2, c_3 for the functions $L_1(n)$, $L_2(n)$ defined in the proof of Theorem 3.8. As is easily seen, these results are sufficient for proving Theorem 3.8.

5. Conclusions and topics for further research. We have shown that there cannot exist minimal-time solutions to the FSSP for our variations 3PATH, g-3PATH, and 3REG if $P \neq NP$, as most researchers believe. The most obvious open problems we leave are the other five variations in section 1.3.2, that is, 2PATH, g-2PATH, 2REG, general undirected networks, and general directed networks. In [19], Kobayashi has shown that a solution to 2PATH is highly unlikely to exist if the two-dimensional path extension problem, 2PEP, is NP-complete. This result can be readily extended to g-2PATH and 2REG. Unfortunately, the computational complexity of 2PEP is unknown and seems difficult to pin down. If there happens to be a polynomial-time solution to 2PEP, it would be interesting from a strictly mathematical point of view, as it is very likely to be nontrivial.

We showed that if 3PEP is decidable in polynomial time, then HAM is decidable in polynomial time. However, this does not imply that 3PEP is NP-complete via a many-one reduction. Our result is only a Turing reduction because the determination of the two values $L_1(n)$ and $L_2(n)$ repeatedly use 3PEP as an oracle. However, it is highly probable that both of the problems 1CUBE and 2CUBE have polynomial-time algorithms, although to prove it will likely require a very tedious and deep analysis of the lengths of paths in cubes. If one can show this, the construction in Theorem 3.8 gives the stronger many-one reduction of HAM to 3PEP.

Another obviously interesting area of exploration would be the opposite direction of Theorem 1.1. What does the complexity-theoretic statement $P = NP$ imply about the existence of a minimal-time solution to the synchronization problem on these networks?

Finally, given that a minimal-time solution to the variation we studied in this paper will most likely not be found (as most researchers believe that $P \neq NP$), it is also an interesting question to ask for approximations. Can we get within a small constant factor of optimal? Given the importance of the synchronization primitive to parallel processing, determining the best approximating factor seems to be a promising and practical avenue of further exploration.

Acknowledgments. We would like to thank Prof. J. Mazoyer for giving us valuable information on minimal-time solutions of variations of FSSP and the reviewers for helpful suggestions for improvements.

REFERENCES

- [1] R. BALZER, *An 8-state minimal time solution to the firing squad synchronization problem*, Information and Control, 10 (1967), pp. 22–42.
- [2] G. DI BATTISTA, P. EADES, R. TAMASSIA, AND I. TOLLIS, *Graph Drawing*, Prentice–Hall, Upper Saddle River, NJ, 1998.

- [3] A. BERTHIAUME, T. BITTNER, L. PERKOVIĆ, A. SETTLE, AND J. SIMON, *Bounding the firing synchronization problem on a ring*, Theoret. Comput. Sci., 320 (2004), pp. 213–228.
- [4] M. CHROBAK AND T. H. PAYNE, *A linear-time algorithm for drawing a planar graph on a grid*, Inform. Process. Lett., 54 (1995), pp. 241–246.
- [5] K. CULIK, *Variations of the firing squad problem and applications*, Inform. Process. Lett., 30 (1989), pp. 153–157.
- [6] H. DE FRAYSSEIX, J. PACH, AND R. POLLACK, *Small sets supporting straight-line embeddings of planar graph*, in Proceedings of the 20th Annual ACM Symposium on Theory of Computing, Chicago, IL, 1988, pp. 426–433.
- [7] H. DE FRAYSSEIX, J. PACH, AND R. POLLACK, *How to draw a planar graph on a grid*, Combinatorica, 10 (1990), pp. 41–51.
- [8] S. EVEN, A. LITMAN, AND P. WINKLER, *Computing with snakes in directed networks of automata*, J. Algorithms, 24 (1997), pp. 158–170.
- [9] M. GAREY AND D. JOHNSON, *Computers and Intractability: A Guide to the Theory of NP-Completeness*, W. H. Freeman, New York, 1979.
- [10] D. GOLDSTEIN AND N. MEYER, *The wake up and report problem is time-equivalent to the firing squad synchronization problem*, in Proceedings of the Thirteenth Annual ACM-SIAM Symposium on Discrete Algorithms, San Francisco, CA, 2002, pp. 578–587.
- [11] E. GOTO, *A Minimal Time Solution of the Firing Squad Problem*, Course Notes for Applied Mathematics 298, Harvard University, Cambridge, MA, 1962, pp. 52–59.
- [12] A. GRASSELLI, *Synchronization of cellular arrays: The firing squad problem in two dimensions*, Information and Control, 28 (1975), pp. 113–124.
- [13] T. JIANG, *The synchronization of nonuniform networks of finite automata*, in Proceedings of the 30th Annual ACM Symposium on Foundations of Computer Science, Duke, NC, 1989, pp. 376–381.
- [14] T. JIANG, *The synchronization of nonuniform networks of finite automata*, Inform. and Comput., 97 (1992), pp. 234–261.
- [15] K. KOBAYASHI, *The firing squad synchronization problem for two-dimensional arrays*, Information and Control, 34 (1977), pp. 177–197.
- [16] K. KOBAYASHI, *The firing squad synchronization problem for a class of polyautomata networks*, J. Comput. System Sci., 17 (1978), pp. 300–318.
- [17] K. KOBAYASHI, *On the minimal firing time of the firing squad synchronization problem for polyautomata networks*, Theoret. Comput. Sci., 7 (1978), pp. 149–167.
- [18] K. KOBAYASHI, *A complexity-theoretical approach to the firing squad synchronization problem*, in Proceedings of JIM '99 (Journée de l'Informatique Messine (Days of Metz Informatics)), “NP-Completeness and Parallelism,” Metz University, Institute of Technology, Metz, France, 1999.
- [19] K. KOBAYASHI, *On time optimal solutions of the firing squad synchronization problem for two-dimensional paths*, Theoret. Comput. Sci., 259 (2001), pp. 129–143.
- [20] K. KOBAYASHI, *A Minimal Time Solution to the Firing Squad Synchronization Problem of Rings with One-Way Information Flow*, Research Report on Information Sciences C-8, Department of Information Sciences, Tokyo Institute of Technology, Tokyo, Japan, 1976.
- [21] S. LATORRE, M. NAPOLI, AND M. PARENTE, *Synchronization of one-way connected processors*, Complex Systems, 10 (1996), pp. 239–255.
- [22] J. MAZOYER, *A six-state minimal time solution to the firing squad synchronization problem*, Theoret. Comput. Sci., 50 (1987), pp. 183–238.
- [23] J. MAZOYER, *An overview of the firing synchronization problem*, in Automata Networks, Lecture Notes in Comput. Sci. 316, Springer-Verlag, Berlin, 1988, pp. 82–94.
- [24] J. MAZOYER, *Synchronization of two interacting finite automata*, Internat. J. Algebra Comput., 5 (1995), pp. 289–308.
- [25] M. MINSKY, *Computation: Finite and Infinite Machines*, Prentice–Hall, Englewood Cliffs, NJ, 1967.
- [26] E. F. MOORE, *Sequential Machines, Selected Papers*, Addison–Wesley, Reading, MA, 1962.
- [27] F. R. MOORE AND G. G. LANGDON, *A generalized firing squad problem*, Information and Control, 12 (1968), pp. 212–220.
- [28] Y. NISHITANI AND N. HONDA, *The firing squad synchronization problem for graphs*, Theoret. Comput. Sci., 14 (1981), pp. 39–61.
- [29] R. OSTROVSKY AND D. WILKERSON, *Faster computation on directed networks of automata*, in Proceedings of the Fourteenth Annual ACM Symposium on Principles of Distributed Computing, Ottawa, ON, Canada, 1995, pp. 38–46.
- [30] F. ROMANI, *Cellular automata synchronization*, Inform. Sci., 10 (1976), pp. 299–318.
- [31] P. ROSENSTIEHL, *Existence d'automates finis capables de s'accorder bien qu'arbitrairement connectées nombreux*, Internat. Comp. Centre Bull., 5 (1966), pp. 245–261.

- [32] P. ROSENSTIEHL, J. R. FIKSEL, AND A. HOLLIGER, *Intelligent graphs: Networks of finite automata capable of solving graph problems*, in Graph Theory and Computing, R. C. Read, ed., Academic Press, New York, 1972, pp. 219–265.
- [33] H. SCHMID AND T. WORSCH, *The firing squad synchronization problem with many generals for one-dimensional CA*, in Proceedings of the 3rd IFIP International Conference on Theoretical Computer Science, Toulouse, France, 2004, pp. 111–124.
- [34] A. SETTLE AND J. SIMON, *Smaller solutions for the firing squad*, Theoret. Comput. Sci., 276 (2002), pp. 83–109.
- [35] I. SHINAHR, *Two- and three-dimensional firing-squad synchronization problem*, Information and Control, 24 (1974), pp. 163–180.
- [36] H. SZWERINSKI, *Time-optimal solution of the firing-squad-synchronization-problem for n -dimensional rectangles with the general at an arbitrary position*, Theoret. Comput. Sci., 19 (1982), pp. 305–320.
- [37] V. I. VARSHAVSKY, V. B. MARAKHOVSKY, AND V. A. PESCHANSKY, *Synchronization of interacting automata*, Math. Systems Theory, 4 (1970), pp. 212–230.
- [38] A. WAKSMAN, *An optimum solution to the firing squad synchronization problem*, Information and Control, 9 (1966), pp. 66–78.
- [39] J.-B. YUNES, *Seven states solutions to the firing squad synchronization problem*, Theoret. Comput. Sci., 127 (1993), pp. 313–332.

STOCHASTIC RELATIONS: CONGRUENCES, BISIMULATIONS AND THE HENNESSY–MILNER THEOREM*

ERNST-ERICH DOBERKAT†

Dedicated to Professor Gerhard Goos

Abstract. The relationship between congruences and bisimulations is investigated for stochastic relations. It is shown that stochastic relations are bisimilar provided they have congruences that generate each other; from this is derived that relations that have isomorphic factor spaces are bisimilar. Stochastic Kripke models are introduced for a modal logic and illustrated for some popular logics. The criterion developed here permits proving for the general case that Kripke models are bisimilar iff they accept exactly the same formulas.

Key words. stochastic relations, stochastic Kripke models, congruences, bisimulations, modal logic, Hennessy–Milner theorem

AMS subject classifications. 68Q85, 08A30, 03B45

DOI. 10.1137/S009753970444346X

1. Introduction and motivation. Larsen and Skou propose in [18] a systematic framework for testing a process against its specification, where a specification may be formulated in formalisms like modal logic, temporal logic, or using process algebra. Testable properties are formulated, and among the logics through which these properties may be formulated a new probabilistic logic is proposed and investigated. The associated process equivalence is called probabilistic bisimulation, and one of the central statements is that—under the technical assumption that the probabilities are not too close to each other—two processes are probabilistically bisimilar iff they satisfy the same formulas in the probabilistic modal logic [18, Theorem 6.4]. This equivalence is dubbed here the *Hennessy–Milner property* for the sake of discussion. The probability theory involved is discrete, so issues of measurability are not really important. Bisimulations are formulated in terms of binary relations on the involved sets.

A little later, Joyal, Nielsen, and Winskel [15] argued that bisimulation between processes can be formulated in terms of spans of open maps between them, hereby pointing to the presentation of models for concurrency through categories of models [15] with the desire for a uniform treatment of all these models. Coalgebras enter the field, and it is shown that the relational formulation of bisimulation for coalgebras for functors based on the category of sets and maps is equivalent with a formulation in terms of spans of projections [25, Example 2.1]. Panangaden’s work [23, 22] suggests that it may be worthwhile to look not only at discrete probability spaces when formulating labeled Markov transition systems, but that nondiscrete probabilities are necessary for modelling concurrency as well. This leads to the question of whether the equivalence of bisimilarity and the acceptance of the same formulas of a modal logic, the Hennessy–Milner property, holds also in a more general probabilistic set-

*Received by the editors May 5, 2004; accepted for publication (in revised form) June 26, 2005; published electronically January 6, 2006. This research was funded in part by *Deutsche Forschungsgemeinschaft*, grant DO 263/8-1, *Algebraische Eigenschaften stochastischer Relationen*.

<http://www.siam.org/journals/sicomp/35-3/44346.html>

†Chair for Software Technology, University of Dortmund, D-44221 Dortmund, Germany (ernst-erich.doberkat@udo.edu).

ting. In fact, Desharnais, Edalat, and Panangaden [6] show that this equivalence is valid when the underlying space is a Polish space (i.e., a completely metrizable and separable topological space) or even an analytic space (i.e., the image of a Polish space under a Borel map) [6]. The elegant solution had a small catch, however: it requires the existence of semipullbacks in the category of stochastic relations. This in turn could only be guaranteed if the transition probabilities are universally measurable, i.e., measurable with respect to an σ -algebra that arises from the Borel sets through an elaborate completion process which adds certain sets of measure zero.

But the natural notion of measurability is Borel measurability, so there remained the question whether the equivalence result can be established for the Borel measurable case, too. This question is not only of conceptual interest: when one wants to apply these results for testing, one definitively does not want to go through a somewhat complex completion process when the Borel sets are readily available. It could be shown in [9, 13] that the semipullback of stochastic relations exists in Polish and in analytic spaces, when the measurable structure is given through the Borel sets. As a consequence of this, the Hennessy–Milner property is established in this scenario, too, under a technical condition that is called *smallness* in [9].

This paper investigates the relationship between modal logics and stochastic relations: it is shown how a general modal logic can be interpreted through a stochastic Kripke model, and that the Hennessy–Milner property holds in general for these stochastic Kripke models. This is dealt with in a somewhat wider context: the argumentation in [6, 9] shows that the equivalence on the states on which the Hennessy–Milner property is based (viz., to be valid for exactly the same formulas) has the property that it is countably generated, leading to the notion of smooth equivalence relations, and to congruences for stochastic relations. Thus we discuss the triangle *congruences*, *bisimulations*, and the *Hennessy–Milner property* in this paper. The following questions are central to the investigations:

1. Can we find an intrinsic criterion for bisimilarity of stochastic relations? “Intrinsic” means that we do not have to appeal to an external instance like a logic for knowing whether or not two relations are bisimilar.

2. Can we formulate stochastic Kripke models for arbitrary modal logics, and does the Hennessy–Milner property still hold?

Both answers are in the positive; we give a sufficient condition for bisimilarity in terms of equivalent congruences. We show that stochastic relations that have isomorphic nontrivial subsystems are bisimilar, and, conversely, that under a topological condition bisimilar relations have nontrivial isomorphic subsystems. The topological condition refers to compactness and continuity, hence it is rather substantial. We conjecture, however, that bisimilarity and the existence of isomorphic factor spaces are equivalent for general analytic spaces. Addressing the second question, we show that bisimilarity and validity for exactly the same formulas are equivalent, so that a general Hennessy–Milner theorem can also be established for the stochastic interpretation of general modal logic. This includes the well-known versions from [6, 9] as special cases. These properties hold under small restrictions (similar to Larsen and Skou’s assumption of not having too-close probabilities) which prevent trivial conditions from creeping in and invalidating the results.

The main cornerstone for these constructions is the existence of semipullbacks for stochastic relations over analytic spaces [13]. We also have a look at smooth equivalence relations over analytic spaces which are of independent interest. We show that these relations have a confluence property that is also crucial in establishing the relationship between congruences and bisimulations for the compact case.

Organization. Section 2 introduces stochastic relations and collects some measure-theoretic results that will be useful. This section also contains the definition of stochastic relations with their morphisms, and of bisimulations. Smooth equivalence relations are introduced formally in section 4, some measure-theoretic machinery associated with them is also developed there, so that we are prepared to enter the world of congruences in section 4. The notion of equivalent congruences is proposed here (after some preparatory work in section 3), it is shown that relations that have equivalent congruences are bisimilar, and an application of this general result to relations with isomorphic factor spaces is formulated and proved. The next two sections give an application to modal logic: section 5 defines Kripke models for a modal similarity type, both nondeterministic and stochastic ones. It is demonstrated in section 5.1 that this can be applied to various well-known logics. Section 5.2 proposes the refinement of a nondeterministic Kripke model through a stochastic one and shows that these refinements exist under some mild topological assumptions. Section 6 is devoted to bisimulations for Kripke models. It is argued that the notion of morphism and, consequently, of bisimulation has to be strengthened for Kripke models, and that then a Hennessy–Milner theorem can be proved: two Kripke models are strongly bisimilar iff they accept exactly the same formulas. Section 7 discusses related work, and section 8 draws some conclusions and gives some indications for further work.

2. Stochastic relations. This section collects in section 2.1 some basic facts and constructions from measure theory for convenience and for later reference. Stochastic relations and the categories we are working with are defined, we provide also a definition of bisimulation.

2.1. Preliminaries.

Polish and analytic spaces. A Polish space (X, \mathcal{G}) is a topological space which is second countable, i.e., which has a countable dense subset, and which is metrizable through a complete metric. A measurable space (X, \mathcal{A}) is a set X with a σ -algebra \mathcal{A} . The Borel sets $\mathcal{B}(X, \mathcal{G})$ for the topology \mathcal{G} is the smallest σ -algebra on X which contains \mathcal{G} . Given two measurable spaces (X, \mathcal{A}) and (Y, \mathcal{B}) , a map $f : X \rightarrow Y$ is \mathcal{A} – \mathcal{B} -measurable whenever

$$f^{-1}[\mathcal{B}] \subseteq \mathcal{A}$$

holds, where

$$f^{-1}[\mathcal{B}] := \{f^{-1}[B] \mid B \in \mathcal{B}\}$$

is the set of inverse images

$$f^{-1}[B] := \{x \in X \mid f(x) \in B\}$$

of elements of \mathcal{B} . Note that $f^{-1}[\mathcal{B}]$ is a σ -algebra, provided \mathcal{B} is one. If the σ -algebras are the Borel sets of some topologies on X and Y , resp., then a measurable map is called *Borel measurable* or simply a *Borel map*. The real numbers \mathbb{R} carry always the Borel structure induced by the usual topology which will usually not be mentioned explicitly when talking about Borel maps.

An *analytic set* $X \subseteq Z$ for a Polish space Z is the image $f[Y]$ of a Polish space Y for some Borel measurable map $f : Y \rightarrow Z$. Endow X with the trace \mathcal{A} of $\mathcal{B}(Z)$ on X , i.e.,

$$\mathcal{A} = \mathcal{B}(Z) \cap X := \{B \cap X \mid B \in \mathcal{B}(Z)\},$$

the elements of which are still called Borel sets. A measurable space (X', \mathcal{A}') which is Borel isomorphic to (X, \mathcal{A}) is called an *analytic space* (a Borel isomorphism is a Borel measurable and bijective map the inverse of which is also Borel measurable).

Inverse systems. The product $(X_1 \times X_2, \mathcal{A}_1 \otimes \mathcal{A}_2)$ of two measurable spaces (X_1, \mathcal{A}_1) and (X_2, \mathcal{A}_2) is the Cartesian product $X_1 \times X_2$ endowed with the σ -algebra

$$\mathcal{A}_1 \otimes \mathcal{A}_2 := \sigma(\{A_1 \times A_2 \mid A_1 \in \mathcal{A}_1, A_2 \in \mathcal{A}_2\}).$$

This is the smallest σ -algebra which contains all the measurable rectangles $A_1 \times A_2$, and it is the smallest σ -algebra \mathcal{E} on $X_1 \times X_2$ which makes the projections $\pi_i : X_1 \times X_2 \rightarrow X_i$ \mathcal{E} - \mathcal{A}_i -measurable for $i = 1, 2$.

Given finite measures μ_i on \mathcal{A}_i , there exists a unique finite measure $\mu_1 \otimes \mu_2$ on $\mathcal{A}_1 \otimes \mathcal{A}_2$ such that

$$(\mu_1 \otimes \mu_2)(A_1 \times A_2) = \mu_1(A_1) \cdot \mu_2(A_2)$$

holds for each $A_1 \in \mathcal{A}_1, A_2 \in \mathcal{A}_2$.

Extending this, let $(\mu_n)_{n \in \mathbb{N}}$ be an inverse system of probabilities on $(X_i, \mathcal{B}(X_i))_{i \in \mathbb{N}}$ for some Polish spaces X_i ; see [24, section V.3]. Thus μ_n is a probability measure on $\mathcal{B}(X_1 \times \dots \times X_n)$ for each natural n such that $\mu_{n+1}(B \times X_{n+1}) = \mu_n(B)$, whenever $B \subseteq X_1 \times \dots \times X_n$ is a Borel set. Then there exists a unique probability measure μ^* on the Borel sets of $\prod_{n \in \mathbb{N}} X_n$ with the property that

$$\mu^* \left(B \times \prod_{m > n} X_m \right) = \mu_n(B)$$

for each Borel set $B \subseteq X_1 \times \dots \times X_n$; see [24, Theorem V.3.1]. μ^* is usually called the *inverse limit* of $(\mu_n)_{n \in \mathbb{N}}$. We will use this construction when illustrating a logic for model checking.

The *direct sum* $(X_1 + X_2, \mathcal{A}_1 + \mathcal{A}_2)$ of the measurable spaces (X_1, \mathcal{A}_1) and (X_2, \mathcal{A}_2) has the direct sum of the underlying base sets as a base set. Then $A \in \mathcal{A}_1 + \mathcal{A}_2$ iff both $A \cap X_1 \in \mathcal{A}_1$ and $A \cap X_2 \in \mathcal{A}_2$ hold. If both spaces are Polish, resp., analytic, their sum is $\mathcal{B}(X_1 + X_2) = \mathcal{B}(X_1) + \mathcal{B}(X_2)$.

We will occasionally talk about the trivial σ -algebra on a set X : the σ -algebra \mathcal{A} is called trivial iff $\mathcal{A} = \{\emptyset, X\}$, hence iff it consists only of the empty set and the entire set X .

When the context is clear, we will write down topological or measurable spaces without their topologies and σ -algebras, resp., and the Borel sets are always understood with respect to the topology under consideration.

Measurable relations. Measurable relations will provide a link between nondeterministic and stochastic systems, as we will see. Let us fix some notations first. Assume that Y and Z is a measurable, resp., Polish space. Consider a set-valued map $R : Y \rightarrow \mathcal{P}(Z)$, (equivalently, a relation $R \subseteq Y \times Z$. We will not distinguish too narrowly between relations and set-valued maps, so that even for a relation R the set $R(x)$ will be defined.) If $R(y)$ always takes closed and nonempty values, and if the (weak) inverse

$$(\exists R)(G) := \{y \in Y \mid R(y) \cap G \neq \emptyset\}$$

is a measurable set, whenever $G \subseteq Z$ is open, then R is called a *measurable relation* on $Y \times Z$. Since Z is Polish, R is a measurable relation iff the strong inverse

$$(\forall R)(F) := \{y \in Y \mid R(y) \subseteq F\}$$

is measurable, whenever $F \subseteq Z$ is closed [14, Theorem 3.5].

We note for later use the representation of weakly measurable relations through measurable selectors (some times called a *Castaing representation*). This representation implies in particular that a weakly measurable set-valued map has a measurable selector. It is established in [27, Theorem 4.2.e].

PROPOSITION 2.1. *Given the Polish spaces Y and Z and a set-valued map $R \subseteq Y \times Z$, then R is weakly measurable iff there exists a sequence $(f_n)_{n \in \mathbb{N}}$ of Borel measurable maps $f_n : Y \rightarrow Z$ such that $\{f_n(y) \mid n \in \mathbb{N}\}$ is dense in $R(y)$ for each $y \in Y$.*

Thus measurable relations have measurable selectors. This will be used when providing a link between nondeterministic and stochastic Kripke models in section 5.2, and it will also help establishing the smoothness of an equivalence relation in section 3.

It is interesting to note that this kind of construction did recently become of interest in modal logics: see, e.g., the discussion of Stone duality in the context of Vietoris topologies on hyperspaces in [17] or the discussion of descriptive general frame in [3, Chapter 5.5].

2.2. Categories of stochastic relations. The intuition behind stochastic relations over, say, a set of states is that each state is assigned not deterministically a new state or a set of possible new states but rather a distribution over the states, indicating with which probability a transition to another state may happen. This is generalized somewhat: first, we will not deal exclusively with probabilities but rather take subprobabilities into account; this is helpful when, e.g., nonterminating computations are modeled. Second, we will not only consider transitions on a space to itself but rather consider something as input-output behavior, so that we get a richer structure (which will pay off when discussing arbitrary modal logics with an operator of arbitrary arity). We will define these stochastic relations now and indicate the basic constructions.

Given two measurable spaces (X, \mathcal{A}) and (Y, \mathcal{B}) , a *stochastic relation* $K : (X, \mathcal{A}) \rightsquigarrow (Y, \mathcal{B})$ is a Borel map from X to the set $\mathfrak{S}(Y, \mathcal{B})$, the latter denoting the set of all subprobability measures on (Y, \mathcal{B}) which carries the *weak*- σ -algebra*. This is the smallest σ -algebra on $\mathfrak{S}(Y, \mathcal{B})$ which renders all maps $\mu \mapsto \mu(D)$ measurable, where $D \in \mathcal{B}$. Hence $K : (X, \mathcal{A}) \rightsquigarrow (Y, \mathcal{B})$ is a *stochastic relation* iff

1. $K(x)$ is a subprobability measure on (Y, \mathcal{B}) for all $x \in X$,
2. $x \mapsto K(x)(D)$ is a measurable map for each measurable set $D \in \mathcal{B}$.

This is a more common formulation of stochastic relations. Note that the *only if*-part follows directly from the definition of the weak*- σ -algebra, and from the fact that the composition of measurable maps is measurable. The *if*-part follows from the observation that a map into $\mathfrak{S}(Y, \mathcal{B})$ for which the setwise evaluations are measurable is weak*-measurable. We will deal usually with stochastic relations between Polish or between analytic spaces. Accordingly, we then call (X, Y, K) a *Polish*, resp., an *analytic object*.

An \mathcal{A} - \mathcal{B} -measurable map $f : X \rightarrow Y$ between the measurable spaces (X, \mathcal{A}) and (Y, \mathcal{B}) induces a map $\mathfrak{S}(f) : \mathfrak{S}(X, \mathcal{A}) \rightarrow \mathfrak{S}(Y, \mathcal{B})$ upon setting

$$\mathfrak{S}(f)(\mu)(D) := \mu(f^{-1}[D])$$

(for $\mu \in \mathfrak{S}(X, \mathcal{A})$, $D \in \mathcal{B}$). It is easy to see that $\mathfrak{S}(f)$ is measurable.

If the stochastic relation $K : (X, \mathcal{A}) \rightsquigarrow (Y, \mathcal{B})$ has the property that $K(x)(Y) = 1$ holds for all $x \in X$, then K is called a *probabilistic relation*. A stochastic relation (X, Y, K) such that Y consists of exactly one point only will be called *degenerate*.

Morphisms. The category **Stoch** has as objects stochastic relations $K = (X, Y, K)$ for measurable spaces X, Y and $K : X \rightsquigarrow Y$. A *morphism*

$$f : K \rightarrow K'$$

between the objects $K = (X, Y, K)$ and $K' = (X', Y', K')$ is a pair $f = (\phi, \psi)$ of surjective measurable maps $\phi : X \rightarrow X'$ and $\psi : Y \rightarrow Y'$ such that

$$K' \circ \phi = \mathfrak{S}(\psi) \circ K$$

holds, i.e., such that the diagram

$$\begin{array}{ccc} X & \xrightarrow{\phi} & X' \\ K \downarrow & & \downarrow K' \\ \mathfrak{S}(Y) & \xrightarrow{\mathfrak{S}(\psi)} & \mathfrak{S}(Y') \end{array}$$

is commutative. The maps underlying morphisms are assumed to be surjective in order to make sure that each target element is actually coming from some source element; technically, surjectivity is required for establishing the existence of semipullbacks; see Theorem 2.2.

The morphisms defined here correspond to *zig-zag-morphisms* in [6, Definition 5.1] when $X = Y$ and $X' = Y'$; because the approach proposed here does not need to work with what is called *simulation morphism* in [6] we need not distinguish the two kinds of morphisms.

We will usually investigate morphisms for stochastic relations based on Polish or analytic spaces. Accordingly, we denote by **P – Stoch** and **A – Stoch** the full subcategories having Polish, resp., analytic objects as their objects.

We note for later use [13, Theorem 2] the existence of semipullbacks in **P – Stoch** and **A – Stoch** (recall that a *semipullback* in a category for a pair of morphisms $f : a \rightarrow c$ and $g : b \rightarrow c$ with the same target is a pair of morphisms $t : p \rightarrow a$ and $s : p \rightarrow b$ with the same source such that $f \circ t = g \circ s$). Since we will refer to the specific shape of the underlying spaces, we indicate its components.

THEOREM 2.2. *Let*

$$K_1 \xrightarrow{f_1} L \xleftarrow{f_2} K_2$$

be morphisms in **A – Stoch** with $f_i = (\phi_i, \psi_i)$, $K_i = (X_i, Y_i, K_i)$, and $L = (A, B, L)$. There exist Polish topologies on

$$\begin{aligned} S &:= \{\langle x_1, x_2 \rangle \in X_1 \times X_2 \mid \phi_1(x_1) = \phi_2(x_2)\}, \\ T &:= \{\langle y_1, y_2 \rangle \in Y_1 \times Y_2 \mid \psi_1(y_1) = \psi_2(y_2)\} \end{aligned}$$

and a stochastic relation $M = (S, T, M)$ such that

1. $p_i : M \rightarrow K_i$ are morphisms in **A – Stoch**, where $p_i = (\pi_{i,S}, \pi_{i,T})$ is constituted by the respective projections, $i = 1, 2$,
2. $f_1 \circ p_1 = f_2 \circ p_2$.

Consequently, both **P – Stoch** and **A – Stoch** have semipullbacks, the underlying object of which is Polish.

Bisimilarity. Bisimilarity is introduced essentially as a span of morphisms [15, 25, 9]. For coalgebras based on the category of sets, this definition agrees with the one through relations originally given by Milner; see [25]. In [7] the authors call a bisimulation what we will introduce as congruence, albeit that paper restricts itself to labeled Markov transition systems, thus technically to families of stochastic relations $S \rightsquigarrow S$ for some state space S . It seems conceptually to be clearer to distinguish spans of morphisms from equivalence relations, thus we make this distinction here. Later we will see some very close connections. We introduce an additional condition which models the tight relationship between bisimilar relations.

DEFINITION 2.3. *The stochastic relations $K = (X, Y, K)$ and $L = (V, W, L)$ are called bisimilar iff there exist a stochastic relation $M = (A, B, M)$, morphisms $f = (\phi, \psi) : M \rightarrow K$, and $g = (\gamma, \delta) : M \rightarrow L$ such that*

1. *the diagram*

$$\begin{array}{ccccc}
 X & \xleftarrow{\phi} & A & \xrightarrow{\gamma} & V \\
 K \downarrow & & M \downarrow & & \downarrow L \\
 \mathfrak{S}(Y) & \xleftarrow{\mathfrak{S}(\psi)} & \mathfrak{S}(B) & \xrightarrow{\mathfrak{S}(\delta)} & \mathfrak{S}(W)
 \end{array}$$

is commutative,

2. *the σ -algebra $\psi^{-1}[\mathcal{B}(Y)] \cap \delta^{-1}[\mathcal{B}(W)]$ is nontrivial, i.e., contains not only \emptyset and B .*

The relation M is called mediating.

The first condition on bisimilarity states that f and g form a span of **Stoch**-morphisms

$$K \xleftarrow{f} M \xrightarrow{g} L;$$

thus we have for each $a \in A, D \in \mathcal{B}(Y), E \in \mathcal{B}(W)$ that the equalities

$$K(\phi(a))(D) = (\mathfrak{S}(\phi) \circ M)(a)(D) = M(a)(\phi^{-1}[D])$$

and

$$L(\psi(a))(E) = (\mathfrak{S}(\psi) \circ M)(a)(E) = M(a)(\psi^{-1}[E])$$

hold. The second condition states that we can find an event $C^* \in \mathcal{B}(B)$ which is common to both K and L in the sense that

$$\psi^{-1}[D] = C^* = \delta^{-1}[E]$$

for some $D \in \mathcal{B}(Y)$ and $E \in \mathcal{B}(W)$ such that both $C^* \neq \emptyset$ and $C^* \neq B$ hold (note that for $C^* = \emptyset$ or $C^* = W$ we can always take the empty or full set, resp.). Given such a C^* with D and E from above we get for each $a \in A$

$$\begin{aligned}
 K(\phi(a))(D) &= M(a)(\psi^{-1}[D]) \\
 &= M(a)(C^*) \\
 &= M(a)(\delta^{-1}[E]) \\
 &= L(\gamma(a))(E);
 \end{aligned}$$

thus the event C^* ties K and L together. Loosely speaking, $\psi^{-1}[\mathcal{B}(Y)] \cap \delta^{-1}[\mathcal{B}(W)]$ can be described as the σ -algebra of common events, which is required to be nontrivial.

Note that without the second condition the two relations K and L , which are strictly probabilistic (i.e., for which the entire space is always assigned probability one), would always be bisimilar: Put $A := X \times V, B := Y \times W$ and set for $\langle x, v \rangle \in A$ as the mediating relation $M(x, v) := K(x) \otimes L(v)$; then the projections will make the diagram commutative. It is also clear that this argument does not work for the sub-probabilistic case. This curious behavior of probabilistic relations is a bit surprising, but it occurs in other situations as well: e.g., it turns out that the full subcategory of probabilistic relations in $\mathbf{A} - \mathbf{Stoch}$ has a final object, while $\mathbf{A} - \mathbf{Stoch}$ itself does not have one; see [12]. The second condition serves to prevent this somewhat anomalous behavior; it is technically not too restrictive, as we will see below.

3. Smooth equivalence relations. Observing systems leads to identifying states, which in turn gives rise to an equivalence relation. Those relations that are suited for our purposes have a countable generator, so that by looking at a countable family of sets it can be decided whether or not two states are equivalent. We call these relations smooth. They may be characterized as the kernels of Borel measurable maps and have technically the remarkable property that factoring does not leave the realm of analytic spaces. We define these relations in this section and carry out some technical constructions which, albeit used for stochastic relations, are discussed best in isolation. The operations concern factoring and an operation we call *spawning*. The latter one permits the description of an equivalence relation and of the associated measurable structure through the machinery of another one. We will need this kind of operation for describing equivalent congruences later on, and we show that isomorphic factor spaces have the property that the relations spawn each other. These results will be taken to bear fruit for stochastic relations in a later section. We show also that in the case of compact base spaces two smooth relations are always related to each other through a kind of confluence property.

Although the results presented here are preparatory in nature, some of them appear to be interesting in their own right.

DEFINITION 3.1. *An equivalence relation ρ on a measurable space (X, \mathcal{A}) is said to be smooth iff there exists a sequence $(A_n)_{n \in \mathbb{N}} \subseteq \mathcal{A}$ such that*

$$x \rho x' \text{ iff } \forall n \in \mathbb{N} : [x \in A_n \Leftrightarrow x' \in A_n].$$

We say that the sequence $(A_n)_{n \in \mathbb{N}}$ determines relation ρ .

A smooth relation ρ on X can be represented as a subset of $X \times X$ as [26, Exercise 5.1.10] shows:

$$\rho = (X \times X) \setminus \bigcap (A_n \times (X \setminus A_n));$$

thus it follows that a smooth equivalence relation is a Borel subset of $X \times X$. The equivalence classes can be expressed in terms of the sequence $(A_n)_{n \in \mathbb{N}}$:

$$[x]_\alpha = \bigcap \{A_n | x \in A_n\} \cap \bigcap \{X \setminus A_n | x \notin A_n\};$$

hence each class is a Borel subset of X .

It is easy to see that ρ is smooth iff $\rho = \ker(f)$ for some $\mathcal{A} - \mathcal{B}(Y)$ -measurable map $f : X \rightarrow Y$ with an analytic space Y . Here

$$\ker(f) := \{\langle x, x' \rangle \mid f(x) = f(x')\}$$

is the kernel of f .

Denote by

$$\mathcal{I}NV(\mathcal{A}, \rho) := \{A \in \mathcal{A} \mid A \text{ is } \rho\text{-invariant}\}$$

the σ -algebra of ρ -invariant measurable sets. Here $A \subseteq X$ is called ρ -invariant iff $A = \bigcup\{[x]_\rho \mid x \in A\}$ holds, thus iff $x \in A$ and $x \rho x'$ together imply $x' \in A$. We will see that smooth equivalence relations with their invariant sets are just the natural kind of equivalence relations compatible with the structure of stochastic relations.

Denote for the equivalence relation ρ on the analytic space X by X/ρ the set of equivalence classes, and let $\eta_\rho : x \mapsto [x]_\rho$ assign to each x its class $[x]_\rho$; denote by $\mathcal{B}(X)/\rho$ the final σ -algebra on X/ρ with respect to the Borel sets $\mathcal{B}(X)$ on X and the natural projection η_ρ .

Smooth equivalence relations arise in a natural fashion from kernels of measurable maps, as we will see in a moment. These relations enjoy the technically interesting property that the factor space $(X/\rho, \mathcal{B}(X)/\rho)$ for an analytic space X and a smooth relation ρ is an analytic space again; cf. [26, Exercise 5.1.14]. In particular, $\mathcal{B}(X/\rho) = \mathcal{B}(X)/\rho$ holds. A little bit more can be said. We introduce a kind of multiplicative operation first. Assume that ρ is a smooth equivalence relation on the analytic space X , and that θ is a smooth equivalence on X/ρ . Define for $x, x' \in X$

$$x (\theta \bullet \rho) x' \Leftrightarrow [x]_\rho \theta [x']_\rho.$$

For later use some useful properties of this operation, and of smooth relations in general, are collected from [11].

LEMMA 3.2. *Let X be an analytic space and ρ a smooth equivalence relation on X .*

1. *Let S be an analytic space, and assume that $f : X \rightarrow S$ is a surjective and Borel measurable map. Then $\ker(f)$ is smooth, and $f^{-1}[\mathcal{B}(S)] = \mathcal{I}NV(\mathcal{B}(X), \ker(f))$.*

2. *The ρ -invariant Borel sets of X are exactly the inverse images of the canonic projection η_ρ , viz., $\mathcal{I}NV(\mathcal{B}(X), \rho) = \eta_\rho^{-1}[\mathcal{B}(X/\rho)]$ holds. If ρ is determined by the sequence $(A_n)_{n \in \mathbb{N}}$ of Borel sets $A_n \subseteq X$, then*

$$\mathcal{I}NV(\mathcal{B}(X), \rho) = \sigma(\{A_n \mid n \in \mathbb{N}\}).$$

3. *If θ is smooth on X/ρ , then $\theta \bullet \rho$ is smooth, and the analytic spaces $X/\theta \bullet \rho$ and $(X/\rho)/\theta$ are Borel isomorphic.*

4. *The following conditions are equivalent for a smooth equivalence relation σ on X :*

- (a) $\rho \subseteq \sigma$,
- (b) *there exists a smooth equivalence relation θ on X/ρ such that $\sigma = \theta \bullet \rho$.*

As a first application, we establish that smoothness is preserved through finite products, and we represent the corresponding invariant Borel sets.

LEMMA 3.3. *Let α and α' be smooth equivalence relations on the analytic spaces X , resp., X' . Define*

$$\langle x_1, x'_1 \rangle (\alpha \times \alpha') \langle x_2, x'_2 \rangle \Leftrightarrow x_1 \alpha x_2 \wedge x'_1 \alpha' x'_2.$$

Then

- 1. $\alpha \times \alpha'$ is a smooth equivalence relation on $X \times X'$,
- 2. $\mathcal{I}NV(\mathcal{B}(X \times X'), \alpha \times \alpha') = \mathcal{I}NV(\mathcal{B}(X), \alpha) \otimes \mathcal{I}NV(\mathcal{B}(X'), \alpha')$.

Proof. Let $\{A_n \mid n \in \mathbb{N}\}$ and $\{A'_n \mid n \in \mathbb{N}\}$ be the generators for $\mathcal{I}NV(\mathcal{B}(X), \alpha)$ and $\mathcal{I}NV(\mathcal{B}(X'), \alpha')$, resp. Then $\{A_n \times A'_n \mid n \in \mathbb{N}\}$ generates

$$\mathcal{I}NV(\mathcal{B}(X), \alpha) \otimes \mathcal{I}NV(\mathcal{B}(X'), \alpha'),$$

and

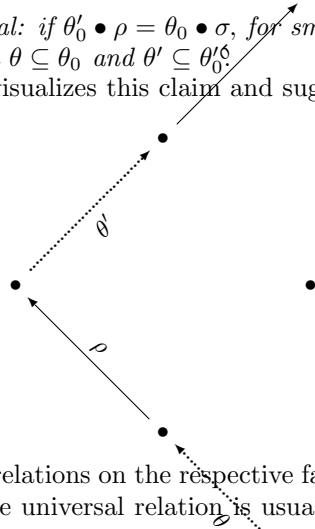
$$\langle x_1, x'_1 \rangle (\alpha \times \alpha') \langle x_2, x'_2 \rangle \Leftrightarrow \forall n \in \mathbb{N} : [\langle x_1, x'_1 \rangle \in A_n \times A'_n \Leftrightarrow \langle x_2, x'_2 \rangle \in A_n \times A'_n]. \quad \square$$

We will establish a confluence property for later use. This property will be helpful for the investigation of the relationship between congruences and bisimulations: we will use this confluence property in a crucial way when it comes to show that bisimilar relations have isomorphic factors.

PROPOSITION 3.4. *Let T be a compact metric space, and assume that $\rho = \ker(\phi), \sigma = \ker(\psi)$ for some continuous maps $\phi : T \rightarrow N, \psi : T \rightarrow N'$ with metric spaces N, N' . There exist smooth equivalence relations θ on T/σ and θ' on T/ρ such that*

1. $\theta' \bullet \rho = \theta \bullet \sigma$,
2. θ and θ' are minimal: if $\theta'_0 \bullet \rho = \theta_0 \bullet \sigma$, for smooth equivalence relations θ_0 on T/σ and θ'_0 on T/ρ , then $\theta \subseteq \theta_0$ and $\theta' \subseteq \theta'_0$.

The following diagram visualizes this claim and suggests the characterization as a confluence property.



Note that the universal relations on the respective factor spaces would satisfy the first condition. But since the universal relation is usually no congruence, it is of no use for our purposes.

The proof will be broken into several parts. Because of part 4 in Lemma 3.2 we will first find a smooth equivalence relation θ on T/σ such that $\rho \subseteq \theta \bullet \sigma$ holds. We will assume through the end of the proof of Proposition 3.4 that T is a compact metric space, and that $\rho = \ker(\phi), \sigma = \ker(\psi)$.

CLAIM 1. *T/σ is a compact metric space when endowed with the final topology for η_σ .*

Proof. Let d' be the metric on N' , and put for $t, t' \in T$

$$D([t]_\sigma, [t']_\sigma) := d'(\psi(t), \psi(t')),$$

then D is a metric on T/σ (since $\sigma = \ker(\psi)$ by assumption). Let \mathcal{T} be the topology on T/σ induced by η_σ , then a set $G \subseteq T/\sigma$ which is D -open is also \mathcal{T} -open. This follows easily from the continuity of ψ . Conversely, let F be \mathcal{T} -closed, and assume that $([t_n]_\sigma)_{n \in \mathbb{N}}$ be a sequence in F such that $D([t_n]_\sigma, [t]_\sigma) \rightarrow 0$, as $n \rightarrow \infty$. Select an arbitrary $x_n \in [t_n]_\sigma$; thus $x_n \in \eta_\sigma^{-1}[F]$. The latter is a closed, hence compact, set. Thus we can find a convergent subsequence (which we take w.l.o.g. the sequence

itself), so that there exists $x^* \in \eta_\sigma^{-1}[F]$ with $x_n \rightarrow x^*$. By the continuity of ψ we may conclude $\psi(x_n) \rightarrow \psi(x^*)$. This implies $x^* \in [t]_\sigma$; thus $[t]_\sigma \in F$. Hence F is also metrically closed, and the topologies coincide. \square

Claim 1 shows among other things that the Borel sets in T/σ come from a compact metric space. This observation will make some arguments easier. When talking about the topology on T/σ , we refer interchangeably to the metric topology and the topology induced by the canonic projection.

CLAIM 2. *Put*

$$\zeta := \{\langle s, s' \rangle \mid s, s' \in T/\sigma, s \times s' \cap \rho \neq \emptyset\}.$$

Then $\zeta \subseteq (T/\sigma)^2$ is reflexive, symmetric, and a closed subset of $(T/\sigma)^2$.

Proof. Since ρ is reflexive and symmetric, ζ is also. Now let $\langle s_n, s'_n \rangle \in \zeta$ be a convergent sequence, say, $s_n \rightarrow s, s'_n \rightarrow s'$. For s_n there exists by the construction of ζ a pair $\langle t_n, t'_n \rangle \in \rho$ with $t_n \in s_n, t'_n \in s'_n$. In particular, $\phi(t_n) = \phi(t'_n)$. Compactness implies the existence of a subsequence $(q(n))_{n \in \mathbb{N}}$ and of elements t, t' such that $t_{q(n)} \rightarrow t, t'_{q(n)} \rightarrow t'$, as $n \rightarrow \infty$. Continuity implies $\langle t, t' \rangle \in \ker(\phi) = \rho$, and $s = [t]_\sigma, s' = [t']_\sigma$. Thus ζ is closed. \square

Now define inductively the n -fold composition of ζ :

$$\begin{aligned} \zeta^{(1)} &:= \zeta, \\ \zeta^{(n+1)} &:= \zeta^{(n)} \circ \zeta, \end{aligned}$$

where \circ denotes the usual relational composition.

The following properties are easily established through a compactness argument using induction on n .

CLAIM 3. *For each $n \in \mathbb{N}$*

1. $\zeta^{(n)} \subseteq T/\sigma$ is closed,
2. if $C \subseteq T/\sigma$ is compact, then the set

$$\exists \zeta^{(n)}(C) = \{s \in T/\sigma \mid \exists s' \in C : \langle s, s' \rangle \in \zeta^{(n)}\}$$

is closed.

CLAIM 4. *The transitive closure θ of ζ is a smooth equivalence relation on T/σ .*

Proof. It is clear from the properties of ζ that θ is an equivalence relation. Smoothness needs to be shown, and we will exhibit a Borel measurable map into a Polish space with θ as its kernel. Write θ as

$$\theta = \bigcup_{n \in \mathbb{N}} \zeta^{(n)},$$

and let $C \subseteq T/\sigma$ be a compact set; then

$$\exists \theta(C) = \{s \in T/\sigma \mid \exists s' \in C : \langle s, s' \rangle \in \theta\}$$

is a measurable subset of T/σ by Claim 3. By Proposition 2.1 we can find a measurable selector w for the set-valued map $s \mapsto \eta_\theta(s)$, and hence a Borel measurable map $w : T/\sigma \rightarrow T/\sigma$ such that $w(s) \in [s]_\theta$ for each $s \in T/\sigma$. Now we know that T/σ is a Polish space, and from $\theta = \ker(w)$ we can infer that θ is smooth. \square

We are in a position now to establish Proposition 3.4.

Proof of Proposition 3.4. 1. It is sufficient for the first part to establish $\rho \subseteq \theta \bullet \sigma$. In fact, let $t \rho t'$; then $[t]_\sigma \times [t']_\sigma \cap \rho \neq \emptyset$. This implies $\langle [t]_\sigma, [t']_\sigma \rangle \in \zeta \subseteq \theta$, which in turn establishes the inclusion and hence the proposition.

2. Now assume that $\theta'_0 \bullet \rho = \theta_0 \bullet \sigma$, for smooth equivalence relations θ_0 on T/σ and θ'_0 on T/ρ . From part 4 of Lemma 3.2 we infer $\sigma \subseteq \theta'_0 \bullet \rho$ and $\rho \subseteq \theta_0 \bullet \sigma$. In order to establish $\theta \subseteq \theta_0$ it is enough to show that $\zeta \subseteq \theta_0$. But if $\langle s, s' \rangle \in \zeta$, we know that $s \times s' \cap \rho \neq \emptyset$. The remark above implies that $s \times s' \cap \theta_0 \bullet \sigma \neq \emptyset$; thus $\langle t, t' \rangle \in \theta_0 \bullet \sigma$ for some $t \in s = [t]_\sigma, t' \in s' = [t']_\sigma$. Consequently, $s \theta_0 s'$ holds. Interchanging the roles of ρ and σ establishes that $\theta' \subseteq \theta'_0$ also holds. \square

For later use we record a property of θ -invariant Borel sets that characterizes these sets in terms of the equivalence relations from which θ is constructed. It gives an easy criterion on invariance and shows that the relation θ is probably not as unpractical as it appears on first sight.

LEMMA 3.5. *Under the assumptions of Proposition 3.4, let $D \subseteq T/\sigma$ be a Borel set, where θ is defined as in Claim 4 as the equivalence relation generated through $\{\langle s, s' \rangle \mid s, s' \in T/\sigma, s \times s' \cap \rho \neq \emptyset\}$. Then these conditions are equivalent:*

1. D is θ -invariant,
2. $\eta_\sigma^{-1}[D] \in \text{INV}(\mathcal{B}(T), \sigma) \cap \text{INV}(\mathcal{B}(T), \rho)$.

Proof. 1. “ $1 \Rightarrow 2$ ”: We know from part 2 of Lemma 3.2 that $\eta_\sigma^{-1}[D] \in \text{INV}(\mathcal{B}(T), \sigma)$, because $D \subseteq T/\sigma$ is a Borel set. Now let $t \in \eta_\sigma^{-1}[D]$ with $t \rho t'$. Then $[t]_\sigma \in D$ and $[t]_\sigma \times [t']_\sigma \cap \rho \neq \emptyset$; thus $\langle [t]_\sigma, [t']_\sigma \rangle \in \theta$, and, since D is θ -invariant, $t' \in \eta_\sigma^{-1}[D]$. Hence $\eta_\sigma^{-1}[D]$ is also ρ -invariant.

2. The implication “ $2 \Rightarrow 1$ ” is established through a routine argument using induction accounting for the construction of θ . \square

As a preparation for the definition of how two smooth relations may be to each other we will have a quick look at how the atoms of a countably generated σ -algebra are characterized through the generators. This representation is established in the proof of [26, section 3.1.15].

LEMMA 3.6. *Let $\mathcal{E} = \sigma(\{E_n \mid n \in \mathbb{N}\})$ be a countably generated σ -algebra over a set E . Define $A^1 := A, A^0 := E \setminus A$ for $A \subseteq E$. Then there exists $\mathbf{F} \subseteq \{0, 1\}^\mathbb{N}$ such that*

$$\left\{ \bigcap_{n \in \mathbb{N}} E_n^{\alpha(n)} \mid \alpha \in \mathbf{F} \right\}$$

are exactly the atoms of \mathcal{E} .

We are ready for a technical definition that permits stating how a smooth equivalence relation is transported through a map between classes in such a way that important properties are maintained. We call this *spawning*. This definition of spawning is still done only for equivalence relations, but it will be later extended to incorporate congruences.

DEFINITION 3.7. *Let α and β be smooth equivalence relations on the analytic spaces X , resp., Y , and assume that $\Upsilon : X/\alpha \rightarrow Y/\beta$ is a map between the equivalence classes. We say that α spawns β via $(\Upsilon, \mathcal{A}_0)$ iff \mathcal{A}_0 is a countable generator of $\text{INV}(\mathcal{B}(X), \alpha)$ such that*

1. \mathcal{A}_0 is closed under finite intersections,
2. $\{\Upsilon_A \mid A \in \mathcal{A}_0\}$ is a generator of $\text{INV}(\mathcal{B}(Y), \beta)$, where $\Upsilon_A := \bigcup \{\Upsilon([x]_\alpha) \mid x \in A\}$,
3. $[x_1]_\alpha = [x_2]_\alpha$ implies the equality of

$$\bigcap \{\Upsilon_A \mid A \in \mathcal{A}_0, x_1 \in A\} \cap \bigcap \{X' \setminus \Upsilon_A \mid A \in \mathcal{A}_0, x_1 \notin A\}$$

and

$$\bigcap \{\Upsilon_A | A \in \mathcal{A}_0, x_2 \in A\} \cap \bigcap \{X' \setminus \Upsilon_A | A \in \mathcal{A}_0, x_2 \notin A\}.$$

Thus if α spawns β , then the measurable structure induced by α on X is all we need for constructing the measurable structure induced by β on Y : the map Υ can be made to carry over the generator \mathcal{A}_0 from $\mathcal{I}NV(\mathcal{B}(X), \alpha)$ to $\mathcal{I}NV(\mathcal{B}(Y), \beta)$ and—in light of Lemma 3.6—to transport the atoms from one σ -algebra to the other. This is of particular interest since the atoms constitute the equivalence classes. Hence α together with Υ and the generator \mathcal{A}_0 is all we may care to know or to learn about β .

The first condition reflects a measure-theoretic precaution: we will need to make sure, e.g., in the construction of the direct sum of stochastic relations that measures are uniquely determined by their values on a generator. This, however, can best be guaranteed if the generator is stable against taking finite intersections. Note that $\Upsilon_{A_1 \cap A_2} = \Upsilon_{A_1} \cap \Upsilon_{A_2}$ also holds, so that closedness under intersections is inherited through Υ .

Whenever we have two smooth equivalence relations such that one spawns the other we obtain on the sum of the underlying spaces a unique smooth relation, the traces of which on the summands are just the given relations. Since we will introduce later on the sum of two relations, this effect will now be studied in greater detail.

LEMMA 3.8. *Let α and β be smooth equivalence relations on the analytic spaces X , resp., Y .*

1. *If α spawns β via $(\Upsilon, \mathcal{A}_0)$, then there exists exactly one smooth equivalence relation γ on $X + Y$ such that $[x]_\gamma \cap X = [x]_\alpha$ and $[x]_\gamma \cap Y = \Upsilon([x]_\alpha)$ hold for all $x \in X$.*
2. *If in addition β spawns α via (Θ, \mathcal{B}_0) , then we have for all $x \in X, y \in Y$ the equivalence*

$$[y]_\beta = \Upsilon([x]_\alpha) \Leftrightarrow [x]_\alpha = \Theta([y]_\beta).$$

Proof. 1. In order to establish property 1, we consider the equivalence relation γ generated from $\{A_n + \Upsilon_{A_n} \mid n \in \mathbb{N}\}$ with $\mathcal{A}_0 = \{A_n \mid n \in \mathbb{N}\}$. Relation γ is evidently smooth and it is uniquely determined through α and β . Let $x \in X$; then we can find $y \in Y$ with $\Upsilon([x]_\alpha) = [y]_\beta$, since Υ maps X/α to Y/β . It is easy to see that

$$\Upsilon([x]_\alpha) = \bigcap \{\Upsilon_{A_n} \mid A_n \in \mathcal{A}_0, x \in A_n\} \cap \bigcap \{Y \setminus \Upsilon_{A_n} \mid A_n \in \mathcal{A}_0, x \notin A_n\}.$$

This establishes part 1.

2. The claim in part 2 follows from the observation that β is the relation on Y which is generated from $\{\Upsilon_{A_n} \mid n \in \mathbb{N}\}$, and that α is the relation on X which is generated from $\{\Theta_{B_n} \mid n \in \mathbb{N}\}$, where $\mathcal{B}_0 = \{B_n \mid n \in \mathbb{N}\}$. \square

We need this construction later on, so we fix it in the following definition.

DEFINITION 3.9. *Let α and β be smooth equivalence relations on the analytic spaces X , resp., Y , and assume that α spawns β via $(\Upsilon, \mathcal{A}_0)$ and β spawns α via (Θ, \mathcal{B}_0) ; the equivalence relation constructed in Lemma 3.8 on $X + Y$ is denoted by $\alpha + \beta$.*

The isomorphism of two factor systems is another illustration of the concept of spawning.

PROPOSITION 3.10. *Let T, T' be analytic spaces with smooth equivalence relations ρ , resp., ρ' . Assume that $\Upsilon : T/\rho \rightarrow T'/\rho'$ is a Borel isomorphism, and let \mathcal{A} be*

a countable generator of $\mathcal{I}NV(\mathcal{B}(X), \rho)$ which is closed under finite intersections. Then ρ spawns ρ' via (Υ, \mathcal{A}) .

Proof. The assumption that the generator \mathcal{A} is closed under finite intersections is easily met: take an arbitrary countable generator \mathcal{A}_0 ; then

$$\{\bigcap \mathcal{F} \mid \mathcal{F} \subseteq \mathcal{A}_0 \text{ is finite}\}$$

is a countable generator which is closed under finite intersections.

1. If $A \in \mathcal{I}NV(\mathcal{B}(T), \rho)$ is a ρ -invariant Borel set, then Υ_A is ρ' -invariant, and it is easily established that

$$\Upsilon_A = \eta_{\rho'}^{-1} [\Upsilon [\eta_{\rho} [A]]]$$

holds. From part 2 in Lemma 3.2 we infer that \mathcal{C}_1 is a generator for $\mathcal{B}(T/\rho)$ where $\mathcal{C}_1 := \eta_{\rho} [\mathcal{I}NV(\mathcal{B}(T), \rho)]$. Consequently, $\{\Upsilon_A \mid A \in \mathcal{A}\}$ is a generator for the ρ' -invariant Borel sets $\mathcal{I}NV(\mathcal{B}(T'), \rho')$, because we may conclude that

$$\begin{aligned} \mathcal{I}NV(\mathcal{B}(T'), \rho') &= \eta_{\rho'}^{-1} [\mathcal{B}(T'/\rho')] \text{ (by Lemma 3.2)} \\ &= \eta_{\rho'}^{-1} [\Upsilon [\mathcal{B}(T'/\rho')]] \text{ (since } \Upsilon \text{ is a Borel isomorphism)} \\ &= \sigma(\{\eta_{\rho'}^{-1} [\Upsilon [C]] \mid C \in \mathcal{C}_1\}) \text{ (by construction of } \mathcal{C}_1\text{)} \\ &= \sigma(\{\Upsilon_A \mid A \in \mathcal{I}NV(\mathcal{B}(T), \rho)\}) \\ &= \sigma(\{\Upsilon_A \mid A \in \mathcal{A}\}) \text{ (since } \mathcal{A} \text{ generates } \mathcal{I}NV(\mathcal{B}(T), \rho)\text{)}. \end{aligned}$$

2. As in the proof of [11, Corollary 1] one shows that

$$\Upsilon([t]_{\rho}) = \bigcap \{\eta_{\rho'} [\Upsilon_A] \mid A \in \mathcal{A}, t \in A\} \cap \bigcap \{\eta_{\rho'} [T' \setminus \Upsilon_A] \mid A \in \mathcal{A}, t \notin A\}.$$

This settles the condition of well-defined atoms and concludes the proof. \square

The proof is technically a bit laborious. The statement, however, will be most useful in permitting us to show that stochastic relations are bisimilar, provided they have isomorphic factors. Working with isomorphisms alone for characterizing bisimilarity may be too strong a condition. In the application to modal logic below we will see that the equivalence relation which is induced on states through having the same logic satisfies the condition on spawning, but it is far from clear in this case whether or not the corresponding factor spaces are Borel isomorphic. Consequently, it seems to be worthwhile to work with the weaker condition.

4. Equivalent congruences. Observing a stochastic system $K : X \rightsquigarrow Y$, pairs with equivalent behavior are identified. This leads to a pair (α, β) of equivalence relations on the inputs X , resp., the outputs Y , with the idea that equivalent inputs lead to equivalent outputs. While equivalent inputs can be described directly through α , the equivalence of outputs requires a description of the level of measurable sets. We argue that a set $B \subseteq Y$ does not distinguish between equivalent outputs iff it is invariant under β , i.e., if $y \in B$ and $y \beta y'$ together imply $y' \in B$. This leads naturally to the notion of a congruence.

DEFINITION 4.1. A congruence $\mathfrak{c} = (\alpha, \beta)$ for the stochastic relation $K = (X, Y, K)$ is a pair of smooth equivalence relations α on X and β on Y such that $K(x)(D) = K(x')(D)$ holds whenever $x \alpha x'$ and D is a β -invariant measurable subset of Y . The congruence \mathfrak{c} is called nontrivial iff the σ -algebra $\mathcal{I}NV(\mathcal{B}(Y), \beta)$ is nontrivial.

Note that each equivalence class of a smooth equivalence relation is an invariant Borel set; thus if we know that $\mathcal{INV}(\mathcal{B}(Y), \beta)$ is trivial for a smooth relation β , then this implies $\beta = Y \times Y$. The exclusion of the universal relation from our discussions is justified through the observation that this relation does not provide us with additional information. If α is a smooth equivalence relation, then $(\alpha, Y \times Y)$ is a congruence, provided $K(x)(Y) = K(x')(Y)$ holds for $x \alpha x'$. But this is not really enough information to work with, because $\mathcal{INV}(\mathcal{B}(Y), Y \times Y) = \{\emptyset, Y\}$, so the invariant sets are not rich enough.

For later use we extend the \bullet -notation to congruences, and order congruences by componentwise refinement. Formally, if $\mathbf{c} = (\alpha, \beta), \mathbf{c}' = (\alpha', \beta')$, then $\mathbf{c} \preceq \mathbf{c}'$ iff both $\alpha \subseteq \alpha'$ and $\beta \subseteq \beta'$ hold.

The next lemma shows that kernels of morphisms and congruences are basically the same thing.

LEMMA 4.2. *If $f : \mathbb{K} \rightarrow \mathbb{K}'$ is a morphism for the stochastic relations \mathbb{K} and \mathbb{K}' , then $\ker(f)$ is a congruence for \mathbb{K} .*

Proof. Let $\mathbb{K} = (X, Y, K)$ and $\mathbb{K}' = (X', Y', K')$ with $f = (\phi, \psi)$. Let $x_1 \in \ker(\phi) \cap x_2$ and $D \subseteq Y$ be a $\ker(\psi)$ -invariant Borel subset of Y . Lemma 3.2 shows that $D = \psi^{-1}[D']$ for some Borel set $D' \subseteq Y'$. Thus we obtain from $f = (\phi, \psi)$ being a morphism

$$\begin{aligned} K(x_1)(D) &= K(x_1)(\psi^{-1}[D']) \\ &= (\mathfrak{S}(\psi) \circ K)(x_1)(D') \\ &= (K' \circ \phi)(x_1)(D') \\ &= K'(\phi(x_1))(D') \\ &= K'(\phi(x_2))(D') \\ &= (K' \circ \phi)(x')(D') \\ &= K(x_2)(D), \end{aligned}$$

the last equality just reversing the argument for the first ones. This establishes the claim. \square

This construction permits introducing factor objects which will be heavily used throughout.

PROPOSITION 4.3. *Let $\mathbf{c} = (\alpha, \beta)$ be a congruence on the stochastic relation $\mathbb{K} = (X, Y, K)$, and define*

$$K_{\alpha, \beta}([x]_{\alpha})(D) := K(x)(\eta_{\beta}^{-1}[D])$$

for $x \in X, D \in \mathcal{B}(Y/\beta)$; then

1. $K_{\alpha, \beta} : X/\alpha \rightsquigarrow Y/\beta$ is a stochastic relation,
2. $\eta_{\mathbf{c}} := (\eta_{\alpha}, \eta_{\beta}) : \mathbb{K} \rightarrow \mathbb{K}/\mathbf{c}$ is a morphism.

We call $\mathbb{K}/\mathbf{c} := (X/\alpha, Y/\beta, K_{\alpha, \beta})$ the factor object (of \mathbb{K} with respect to \mathbf{c}).

Proof. 1. Given $D \in \mathcal{B}(Y/\beta)$, $\eta_{\beta}^{-1}[D]$ is an invariant Borel set; thus $K(x)(\eta_{\beta}^{-1}[D])$ does depend only on the α -class of $x \in X$. Consequently, $K_{\alpha, \beta}$ is well defined.

2. $K_{\alpha, \beta} : X/\alpha \rightsquigarrow Y/\beta$ is a stochastic relation. In fact, it is plain that $K_{\alpha, \beta}([x]_{\alpha})$ is a subprobability measure on $\mathcal{B}(Y/\beta)$, so it remains to be shown that $t \mapsto K_{\alpha, \beta}(t)(D)$ is a $\mathcal{B}(X/\alpha)$ -measurable map for each $D \in \mathcal{B}(Y/\beta)$. Fix such a D and a Borel set $F \subseteq \mathbb{R}$; then

$$F_D := \{x \in X \mid K(x)(\eta_{\beta}^{-1}[D]) \in F\}$$

is a Borel set in X , and since $\eta_\beta^{-1}[D]$ is β -invariant, F_D is α -invariant with

$$\{t \in X/\alpha \mid K_{\alpha,\beta}(t)(D) \in F\} = \eta_\alpha[F_D] \in \mathcal{B}(X/\alpha)$$

by Lemma 3.2. This establishes measurability.

3. The construction of $K_{\alpha,\beta}$ yields $K_{\alpha,\beta} \circ \eta_\alpha = \mathfrak{S}(\eta_\beta) \circ K$; hence η_c is a morphism. \square

An important instance of congruences and factor spaces is furnished through equivalent congruences.

DEFINITION 4.4. Let $K = (X, Y, K)$ and $K' = (X', Y', K')$ be Polish objects with congruences $\mathfrak{c} = (\alpha, \beta)$ and $\mathfrak{c}' = (\alpha', \beta')$, respectively.

1. Call \mathfrak{c} proportional to \mathfrak{c}' (symbolically $\mathfrak{c} \propto \mathfrak{c}'$) iff α spawns α' via $(\Upsilon, \mathcal{A}_0)$ and β spawns β' via (Θ, \mathcal{B}_0) such that

$$\forall x \in X \forall x' \in \Upsilon([x]_\alpha) \forall B \in \mathcal{B}_0 : K(x)(B) = K'(x')(\Theta_B).$$

2. Call these congruences equivalent iff both $\mathfrak{c} \propto \mathfrak{c}'$ and $\mathfrak{c}' \propto \mathfrak{c}$ hold.

Thus equivalent congruences behave in exactly the same way. The same behavior is exhibited on each atom, i.e., equivalence class, as far as the input is concerned, and the respective invariant output sets. It becomes visible now that a characterization of equivalent behavior through congruences requires the double face of congruences: it is certainly necessary to use the equivalence relation on the input spaces, but since the behavior on the output spaces is modeled through probabilities, we also need the invariant Borel sets for a characterization.

We will now show how equivalent congruences on stochastic relations give rise to a factor object built on their sum. This construction will be of use in Proposition 4.5 for investigating the bisimilarity of stochastic relations.

Assume that \mathfrak{c} and \mathfrak{c}' are equivalent congruences on the Polish objects $K = (X, Y, K)$ and $K' = (X', Y', K')$, respectively. Construct for K and K' the direct sum

$$K \oplus K' := (X + X', Y + Y', K \oplus K'),$$

where the only nonobvious construction is $K \oplus K'$: put for the Borel set $E \subseteq Y + Y'$

$$(K \oplus K')(z)(E) := \begin{cases} K(z)(E \cap Y) & \text{if } z \in X, \\ K'(z)(E \cap Y') & \text{if } z \in X'. \end{cases}$$

Then clearly $K \oplus K' : X + X' \rightsquigarrow Y + Y'$. Define on $X + X'$, resp., $Y + Y'$, the σ -algebras

$$\begin{aligned} \mathcal{G} &:= \{C + C' \mid C \in \text{INV}(\mathcal{B}(X), \alpha), C' \in \text{INV}(\mathcal{B}(X'), \alpha')\}, \\ \mathcal{H} &:= \{D + D' \mid D \in \text{INV}(\mathcal{B}(Y), \beta), D' \in \text{INV}(\mathcal{B}(Y'), \beta')\}. \end{aligned}$$

Then \mathcal{G} and \mathcal{H} are countable generated sub- σ -algebras of the respective Borel sets. Because the σ -algebras in question are countably generated, so is their sum, and because the congruences are equivalent, we claim that $z \sim (\alpha + \alpha') z'$ implies that $(K \oplus K')(z)(F) = (K \oplus K')(z')(F)$ holds for all $F \in \mathcal{H}$. To establish this, fix $z \in X, z' \in X'$, and consider

$$\mathcal{S} := \{F \in \mathcal{H} \mid (K \oplus K')(z)(F) = (K \oplus K')(z')(F)\}.$$

Since the congruences are equivalent, this is a σ -algebra containing the generator $\{D_n + \Theta_{D_n} \mid n \in \mathbb{N}\}$, where β spawns β' via $(\Theta, \{D_n \mid n \in \mathbb{N}\})$. Since the generator is closed under finite intersections, measures are uniquely determined. This implies $\mathcal{H} \subseteq \sigma(\mathcal{S})$, and thus $\mathcal{H} = \mathcal{S}$. Consequently,

$$\begin{aligned} \mathcal{G} &= \text{INV}(\mathcal{B}(X + X'), \alpha + \alpha'), \\ \mathcal{H} &= \text{INV}(\mathcal{B}(Y + Y'), \beta + \beta'), \end{aligned}$$

and $\mathbf{c} + \mathbf{c}' := (\alpha + \alpha', \beta + \beta')$ is a congruence on $\mathbf{K} \oplus \mathbf{K}'$.

The factor object

$$(\mathbf{K} \oplus \mathbf{K}') / (\mathbf{c} + \mathbf{c}')$$

constructed in this way will be investigated more closely in Proposition 4.5 below when we establish that \mathbf{K} and \mathbf{K}' are bisimilar, provided they have equivalent nontrivial congruences.

Equivalent congruences give rise to bisimilar stochastic relations. This is a rather far-reaching generalization of the by now well-known characterization of bisimilarity of labeled Markov transition systems through mutually equivalent states. Note that this is an intrinsic characterization of bisimilarity: we investigate the relations and their congruences on their own, but we do not need an external instance (like a logic) to determine bisimilarity.

PROPOSITION 4.5. *If \mathbf{c}_i are equivalent nontrivial congruences on the Polish objects \mathbf{K}_i for $i = 1, 2$, then \mathbf{K}_1 and \mathbf{K}_2 are bisimilar.*

Proof. 1. Let $\mathbf{K}_i = (X_i, Y_i, K_i)$ and $\mathbf{c}_i = (\alpha_i, \beta_i)$ for $i = 1, 2$. Construct the sum $\mathbf{K}_1 \oplus \mathbf{K}_2$ as above, and let (κ_i, λ_i) be the corresponding injections, which are, however, not morphisms. Let

$$(\eta_{\alpha_1 + \alpha_2}, \eta_{\beta_1 + \beta_2}) : \mathbf{K}_1 \oplus \mathbf{K}_2 \rightarrow (\mathbf{K}_1 \oplus \mathbf{K}_2) / (\mathbf{c}_1 + \mathbf{c}_2)$$

be the canonical injection; then

$$(\eta_{\alpha_1 + \alpha_2} \circ \kappa_i, \eta_{\beta_1 + \beta_2} \circ \lambda_i)$$

constitutes a morphism $\mathbf{K}_i \rightarrow (\mathbf{K}_1 \oplus \mathbf{K}_2) / (\mathbf{c}_1 + \mathbf{c}_2)$, as will be shown now. Surjectivity has to be established, and we have to show that the σ -algebra of common events is nontrivial.

2. Each equivalence class $a \in (X_1 + X_2) / (\alpha_1 + \alpha_2)$ can be represented as

$$a = [x_1]_{\alpha_1} + [x_2]_{\alpha_2}$$

for some suitably chosen $x_1 \in X_1, x_2 \in X_2$. Similarly, each equivalence class $b \in (Y_1 + Y_2) / (\beta_1 + \beta_2)$ can be written as

$$b = [y_1]_{\beta_1} + [y_2]_{\beta_2}$$

for some $y_1 \in Y_1, y_2 \in Y_2$. Conversely, the sum of classes is a class again. This follows from Lemma 3.8.

3. The semipullback of the pair of morphisms with a joint target constructed in the first step exists by Theorem 2.2. It is a Polish object (A, B, M) , where

$$B = \{\langle y_1, y_2 \rangle \in Y_1 \times Y_2 \mid [y_1]_{\beta_1 + \beta_2} = [y_2]_{\beta_1 + \beta_2}\}.$$

Since \mathbf{c} is nontrivial, we can find an invariant Borel set $D \in \mathcal{I}NV(\mathcal{B}(Y_1), \beta_1)$ with $\emptyset \neq D \neq Y_1$. Assume that β_1 spawns β_2 via $(\Theta, \{D_n \mid n \in \mathbb{N}\})$; then $\emptyset \neq \Theta_D \neq Y_2$ also holds. Because D is β_1 -invariant,

$$\begin{aligned} \pi_{1,Y_1}^{-1}[D] &= \{\langle y_1, y_2 \rangle \mid y_1 \in D\} \\ &= \{\langle y_1, y_2 \rangle \mid y_2 \in \Theta_D\} \\ &= \pi_{2,Y_2}^{-1}[\Theta_D]; \end{aligned}$$

thus

$$\pi_{1,Y_1}^{-1}[D] \in \pi_{1,Y_1}^{-1}[\mathcal{B}(Y_1)] \cap \pi_{2,Y_2}^{-1}[\mathcal{B}(Y_2)],$$

and we are done once it is shown that $\pi_{1,Y_1}^{-1}[D] \neq B$. Since $D \neq Y_1$ is invariant, there exists y_1 with

$$[y_1]_{\beta_1+\beta_2} \cap D = [y_1]_{\beta_1} \cap D = \emptyset.$$

Let $[y_2]_{\beta_2} := \Theta([\psi_1]_{\beta_1})$; then

$$[y_2]_{\beta_1+\beta_2} \cap \Theta_D = [y_2]_{\beta_2} \cap \Theta_D = \emptyset.$$

Consequently, $\langle y_1, y_2 \rangle \in B \setminus \pi_{1,Y_1}^{-1}[D]$. This shows that $\pi_{1,Y_1}^{-1}[\mathcal{B}(Y_1)] \cap \pi_{2,Y_2}^{-1}[\mathcal{B}(Y_2)]$ is nontrivial. \square

The proof's strategy is to make sure that the classes associated with the congruences are distributed evenly among the summands in the sense that each class in the sum is the sum of appropriate classes. This then implies that we can construct surjective maps, and from them morphisms through some general mechanisms. This idea works, in particular, with isomorphic factor spaces.

PROPOSITION 4.6. *Let K and K' be analytic objects such that K/\mathbf{c} is isomorphic to K'/\mathbf{c}' for some nontrivial congruences \mathbf{c} and \mathbf{c}' . Then K and K' are bisimilar.*

Proof. Let $K = (X, Y, K)$ with $\mathbf{c} = (\alpha, \beta)$, and similarly for K' and \mathbf{c}' . Assume that $\mathbf{f} = (\Phi, \Psi)$ is the isomorphism $K/\mathbf{c} \rightarrow K'/\mathbf{c}'$ which is composed of the Borel isomorphisms $\Phi : X/\alpha \rightarrow X'/\alpha'$ and $\Psi : Y/\beta \rightarrow Y'/\beta'$. Moreover, let \mathcal{A} and \mathcal{B} be countable generators of $\mathcal{I}NV(\mathcal{B}(X), \alpha)$ and $\mathcal{I}NV(\mathcal{B}(Y), \beta)$ which are closed under finite intersections. We know from Proposition 3.10 that α spawns α' via (Φ, \mathcal{A}) , and that β spawns β' via (Ψ, \mathcal{B}) . Hence we have to establish for each $x \in X, x' \in \Phi([x]_\alpha)$ and for each β -invariant Borel subset $B \subseteq Y$ that

$$K(x)(B) = K'(x')(\Psi_B)$$

holds. This will imply $\mathbf{c} \propto \mathbf{c}'$. Interchanging the roles of \mathbf{c} and \mathbf{c}' then will yield the result.

1. Given $B \in \mathcal{I}NV(\mathcal{B}(Y), \beta)$ we know from Lemma 3.2 that we can find a Borel set $B_1 \in \mathcal{B}(Y/\beta)$ such that $B = \eta_\beta^{-1}[B_1]$. Since Ψ is a Borel isomorphism, we find $B_2 \in \mathcal{B}(Y'/\beta')$ with $B_1 = \Psi^{-1}[B_2]$. A routine calculation shows that $\Psi_B = \eta_{\beta'}^{-1}[B_2]$. Now assume that $x \in X, x' \in \Phi([x]_\alpha)$; then the following chain of equations is obtained from the argumentation above, and from the assumption that \mathbf{f} is an isomorphism.

$$\begin{aligned} K(x)(B) &= K(x)(\eta_\beta^{-1}[\Psi^{-1}[B_2]]) \\ &= K_{\alpha,\beta}([x]_\alpha)(\Psi^{-1}[B_2]) \\ &= K'_{\alpha',\beta'}(\Phi([x]_\alpha))(B_2) \\ &= K'(x')(\eta_{\beta'}^{-1}[B_2]) \\ &= K'(x')(\Psi_B). \end{aligned}$$

This establishes the desired relation $c \propto c'$ and completes the proof. \square

Thus isomorphic factor spaces make sure that the relations are bisimilar. The natural question is whether or not the converse also holds: given bisimilar relations, do they have isomorphic factor spaces? A first step toward an answer is shown in the following proposition.

PROPOSITION 4.7. *If the Polish objects K and K' are bisimilar such that the mediating object is compact with continuous morphisms, then K and K' have isomorphic nondegenerate factor spaces.*

Proof. 1. Let

$$K \xleftarrow{f} M \xrightarrow{f'} K'$$

be the span of morphisms constituting bisimilarity. Because K is isomorphic to $M/\ker(f)$ by [11, Corollary 3], we may restrict our attention to factors of M . Thus we assume that $K = M/c, K' = M/c'$, where both c and c' are the kernels of continuous morphisms. Suppose that we can find congruences d and d' such that $d \bullet c = d' \bullet c'$. Then

$$\begin{aligned} K/d &= (M/c)/d \\ &\cong M/d \bullet c \text{ (by [11, Proposition 2])} \\ &= M/d' \bullet c' \\ &= (M/c')/d' \\ &\cong K'/d' \end{aligned}$$

(\cong indicating isomorphism), and we are done, provided K/d is shown to be nondegenerate, or, equivalently, d not to have the universal relation as its second component. When looking for suitable congruences d and d' it is in view of [11, Corollary 4] sufficient to find a congruence $d' = (\gamma, \delta)$ with $c \preceq d' \bullet c'$ for the given congruences c and c' , and δ is not universal.

2. Assume $M = (X, Y, M)$, and suppose $c = (\alpha, \beta), c' = (\alpha', \beta')$. We know that there exist smooth equivalence relations γ and δ with $\alpha \subseteq \gamma \bullet \alpha'$ and $\beta \subseteq \delta \bullet \beta'$; moreover, we know for a δ -invariant Borel subset $D \in \text{INV}(Y/\beta', \delta)$ that

$$\eta_{\beta'}^{-1}[D] \in \text{INV}(\mathcal{B}(Y), \beta) \cap \text{INV}(\mathcal{B}(Y), \beta').$$

This was shown in Proposition 3.4 and Lemma 3.5.

We show that $d = (\gamma, \delta)$ is a congruence; thus we have to show that $K_{\alpha', \beta'}(s)(D) = K_{\alpha', \beta'}(s')(D)$, whenever D is a δ -invariant Borel subset of Y/β' and $s \gamma s'$.

Assume first that

$$\langle s, s' \rangle \in \gamma_0 := \{ \langle t, t' \rangle \mid t, t' \in X/\alpha', t \times t' \cap \alpha \neq \emptyset \}.$$

Then we can find $\langle x, x' \rangle \in \alpha$ such that $s = [x]_{\alpha'}, s' = [x']_{\alpha'}$, and $[x]_{\alpha} = [x']_{\alpha}$. Thus we obtain from D 's invariance properties

$$\begin{aligned} K_{\alpha', \beta'}(s)(D) &= K(x)(\eta_{\beta'}^{-1}[D]) \\ &= K(x')(\eta_{\beta'}^{-1}[D]) \\ &= K_{\alpha', \beta'}(s')(D). \end{aligned}$$

This means that the assertion is true for all $\langle s, s' \rangle \in \gamma_0$.

Now consider

$$\hat{\gamma} := \{ \langle t, t' \rangle \mid t, t' \in X/\alpha', K_{\alpha', \beta'}(t)(D) = K_{\alpha', \beta'}(t')(D) \};$$

then $\hat{\gamma}$ is an equivalence relation which contains γ_0 , and, consequently, it contains γ , as the construction of γ as the transitive closure of γ_0 shows (see Claim 4).

3. Since M/c and M'/c' are bisimilar, we can find $F \in \text{INV}(\mathcal{B}(Y), \beta) \cap \text{INV}(\mathcal{B}(Y), \beta')$ with $\emptyset \neq F \neq Y$ (this is so since, e.g., $\text{INV}(\mathcal{B}(Y), \beta) = \eta_\beta^{-1}[\mathcal{B}(Y/\beta)]$ by Lemma 3.2, part 1). Now minimality of the construction leading to Proposition 3.4 enters the game: from Lemma 3.5 we infer that

$$\eta_{\beta'}^{-1}[\text{INV}(\mathcal{B}(Y/\beta'), \delta)] = \text{INV}(\mathcal{B}(Y), \beta) \cap \text{INV}(\mathcal{B}(Y), \beta')$$

holds; thus we can find $F_0 \in \text{INV}(\mathcal{B}(Y/\beta'), \delta)$ with $\emptyset \neq F_0 \neq Y/\beta'$. Consequently, δ is not universal and we are done. \square

Summarizing, we have established the following characterization of bisimilarity through congruences.

THEOREM 4.8. *Consider for analytic objects K and K' the following statements.*

1. *There exist nontrivial congruences c and c' on K , resp., K' , such that K/c and K'/c' are isomorphic.*
2. *K and K' are bisimilar.*

Then $1 \Rightarrow 2$ holds always, and $2 \Rightarrow 1$ holds in the case that the mediating object is compact and the associated morphisms are continuous.

This is an intrinsic characterization of bisimilarity through congruences, because we can look at the stochastic relations and say that they are bisimilar without having to look at an external instance (like the sentences of a modal logic). It would be most valuable to lift the rather strong condition on compactness. The proofs given above, in particular in section 3, are Claims 1 through 4 relying on compact spaces via the possibility to extract from each sequence a converging subsequence (hence on sequential compactness, to be specific). Otherwise smoothness cannot be guaranteed, but smoothness is crucial since it makes sure that the factor space is analytic.

Conjecture. The characterization of bisimilarity through isomorphic factor spaces is valid for all stochastic relations over analytic spaces.

5. Interpretations of modal logic. We have so far established a criterion for bisimilarity through equivalent congruences and discussed bisimilarity in terms of isomorphic factor spaces. The rest of the paper will apply this to modal logic. This section defines the logic we will be working with, and Kripke models are defined in their usual nondeterministic and stochastic versions, together with their satisfaction relation. In section 5.1 some examples are given in order to demonstrate how probabilistic models are defined for specific logics, and in section 5.2 we relate nondeterministic to stochastic interpretations by introducing probabilistic refinements.

Let P be a countable set of propositional letters which is fixed throughout; $O \neq \emptyset$ is a set of modal operators. Following [3], $\tau = (O, \rho)$ is called a *modal similarity type* iff $O \neq \emptyset$, and if $\rho : O \rightarrow \mathbb{N}$ is a map, assigning each modal operator Δ its arity $\rho(\Delta) \geq 1$. We will not deal with modal operators of arity zero, since as modal constants they do not have to be dealt with in an interpretation. The similarity type τ will be fixed.

We define three modal languages based on τ and P . The formulas of the *basic modal language* $\mathfrak{Mod}_b(\tau, P)$ are given by the syntax

$$\varphi ::= p \mid \top \mid \varphi_1 \wedge \varphi_2 \mid \neg\varphi \mid \Delta(\varphi_1, \dots, \varphi_{\rho(\Delta)}),$$

where $p \in P$. Omitting negation defines the formulas in the *negation free basic modal language* $\mathfrak{Mod}_1(\tau, P)$. Finally, the *extended modal language* $\mathfrak{Mod}_s(\tau, P)$ is defined through the syntax

$$\varphi ::= p \mid \top \mid \varphi_1 \wedge \varphi_2 \mid \neg\varphi \mid \Delta_q(\varphi_1, \dots, \varphi_{\rho(\Delta)}),$$

where $q \in \mathbb{Q} \cap [0, 1]$ is a rational number, and $p \in P$ is a propositional letter.

A *nondeterministic τ -Kripke model* $\mathcal{R} = (S, R_\tau, V)$ consists of a state space S , a family $R_\tau = ((R_\Delta)_{\Delta \in O})$ of set-valued maps $R_\Delta : S \rightarrow \mathcal{P}(S^{\rho(\Delta)})$, and a set-valued map $V : P \rightarrow \mathcal{P}(S)$.

The satisfaction relation for a nondeterministic τ -Kripke model \mathcal{R} is defined as usual for $\mathfrak{Mod}_b(\tau, P)$:

- (i) $\mathcal{R}, s \models p \Leftrightarrow s \in V(p)$,
- (ii) $\mathcal{R}, s \models \neg\varphi \Leftrightarrow \mathcal{R}, s \not\models \varphi$,
- (iii) $\mathcal{R}, s \models \varphi_1 \wedge \varphi_2 \Leftrightarrow \mathcal{R}, s \models \varphi_1$ and $\mathcal{R}, s \models \varphi_2$,
- (iv) $\mathcal{R}, s \models \Delta(\varphi_1, \dots, \varphi_{\rho(\Delta)}) \Leftrightarrow \exists \langle s_1, \dots, s_{\rho(\Delta)} \rangle \in R_\Delta(s) : \mathcal{R}, s_i \models \varphi_i$ for $1 \leq i \leq \rho(\Delta)$.

Denote by $\llbracket \varphi \rrbracket_{\mathcal{R}} := \{s \in S \mid \mathcal{R}, s \models \varphi\}$ the set of states for which formula φ is valid, and by

$$Th_{\mathcal{R}}(s) := \{\varphi \in \mathfrak{Mod}_b(\tau, P) \mid \mathcal{R}, s \models \varphi\}$$

the theory of state s in \mathcal{R} .

An easy calculation shows that

$$\mathcal{R}, s \models \Delta(\varphi_1, \dots, \varphi_{\rho(\Delta)}) \Leftrightarrow R_\Delta(s) \cap \llbracket \varphi_1 \rrbracket_{\mathcal{R}} \times \dots \times \llbracket \varphi_{\rho(\Delta)} \rrbracket_{\mathcal{R}} \neq \emptyset.$$

In analogy, a *stochastic τ -Kripke model* $\mathcal{K} = (S, K_\tau, V)$ has a state space S which is endowed with a σ -algebra, a family $K_\tau = (K_\Delta)_{\Delta \in O}$ of stochastic relations $K_\Delta : S \rightsquigarrow S^{\rho(\Delta)}$, and a set-valued map $V : P \rightarrow \mathcal{P}(S)$ such that $V(p)$ is always a measurable set (we leave the σ -algebra on the state space anonymous to avoid cluttering the notation; note that $S^{\rho(\Delta)}$ carries the product σ -algebra).

The interpretation of formulas in $\mathfrak{Mod}_s(\tau, P)$ for a stochastic τ -Kripke model \mathcal{K} is fairly straightforward, the interesting case arising when a modal operator is involved:

$$\mathcal{K}, s \models \Delta_q(\varphi_1, \dots, \varphi_{\rho(\Delta)})$$

holds iff there exists measurable subsets $A_1, \dots, A_{\rho(\Delta)}$ of S such that $\mathcal{K}, s_i \models \varphi_i$ holds for all $s_i \in A_i$ for $1 \leq i \leq \rho(\Delta)$, and

$$K_\Delta(s)(A_1 \times \dots \times A_{\rho(\Delta)}) \geq q.$$

Arguing from the point of view of state transition systems, this interpretation of validity reflects that upon the move indicated by Δ_q , a state s satisfies $\Delta_q(\varphi_1, \dots, \varphi_{\rho(\Delta)})$ iff we can find states s_i satisfying φ_i with a K_Δ -probability exceeding q . Note that the usual operators Δ and ∇ are replaced by a whole spectrum of operators Δ_q which permit a finer and probabilistically more adequate notion of satisfaction (see [18]).

Again, let $\llbracket \varphi \rrbracket_{\mathcal{K}}$ be the set of all states for which $\varphi \in \mathfrak{Mod}_s(\tau, P)$ is satisfied under \mathcal{K} , and

$$Th_{\mathcal{K}}(s) := \{\varphi \in \mathfrak{Mod}_s(\tau, P) \mid \mathcal{K}, s \models \varphi\}$$

the state’s theory.

It turns out that the sets $\llbracket \varphi \rrbracket_{\mathcal{K}}$ are measurable, so that they may be used as arguments for the stochastic relations we are working with.

LEMMA 5.1. $\llbracket \varphi \rrbracket_{\mathcal{K}}$ is a measurable subset of S for each $\varphi \in \mathfrak{Mod}_s(\tau, P)$.

Proof. The proof proceeds by induction on φ . If $\varphi = p \in P$, then $\llbracket \varphi \rrbracket_{\mathcal{K}} = V(p)$ holds, which is measurable by assumption. Since the measurable sets are closed under complementation and intersection, the only interesting case is again the one in which a modal operator is involved. Since

$$\llbracket \Delta_q(\varphi_1, \dots, \varphi_{\rho(\Delta)}) \rrbracket_{\mathcal{K}} = \{s \in S \mid K_{\Delta}(s)(\llbracket \varphi_1 \rrbracket_{\mathcal{K}} \times \dots \times \llbracket \varphi_{\rho(\Delta)} \rrbracket_{\mathcal{K}}) \geq q\},$$

the assertion follows from the induction hypothesis and the fact that K_{Δ} is a stochastic relation. \square

As in the case of stochastic relations, we need to exclude trivial cases.

DEFINITION 5.2. A τ -Kripke model \mathcal{K} with state space S is called degenerate iff $\llbracket \varphi \rrbracket_{\mathcal{K}} = S$ or $\llbracket \varphi \rrbracket_{\mathcal{K}} = \emptyset$ holds for each formula $\varphi \in \mathfrak{Mod}_s(\tau, P)$.

Hence a degenerate model does not carry useful information.

5.1. Examples. We show how some popular logics may be interpreted through Kripke models, indicating that specific logics require specific probabilistic arguments. But first we indicate that each stochastic relation may be “trained” to interpret a modal logic simply by interpreting the subformulas of a compound formula as stochastically independent. Then we introduce the well-known logic associated with labeled transition systems. This example is of historic significance, given the seminal work of Larsen and Skou [18]. It is shown also how the basic temporal language may be interpreted stochastically by reversing a relation, and arrow logic as a popular logic modelling simple programming constructs is interpreted through a simple transformation of a distribution. The last example leaves the realm of modal logics somewhat by tackling a logic that is used for model checking. It will be shown how a stochastic relation generates path probabilities (through an inverse limit construction), and how this can be made use of for a probabilistic interpretation of a simple kind of tree logic.

A stochastic relation on the state space induces a stochastic τ -Kripke model. This is illustrated through the following example.

Example 5.1. Let $K : S \rightsquigarrow S$ be a stochastic relation on the state space S , and define for $s \in S$ and for the modal operator Δ

$$K_{\Delta}(s) := \bigotimes_{i=1}^{\rho(\Delta)} K(s).$$

Then $K_{\Delta} : S \rightsquigarrow S^{\rho(\Delta)}$ is a stochastic relation. If $V(p) \subseteq S$ is a measurable subset of S , then

$$\mathcal{K}_{K,V} := (S, (K_{\Delta})_{\Delta \in O}, V)$$

is a stochastic τ -Kripke model such that

$$\mathcal{K}_{K,V}, s \models \Delta_q(\varphi_1, \dots, \varphi_{\rho(\Delta)}) \Leftrightarrow K(s)(\llbracket \varphi_1 \rrbracket_{\mathcal{K}_{K,V}}) \cdots K(s)(\llbracket \varphi_{\rho(\Delta)} \rrbracket_{\mathcal{K}_{K,V}}) \geq q.$$

Thus the arguments to each modal operator are stochastically independent.

Example 5.2. Suppose that L is a countable alphabet of actions. Each action $a \in L$ is associated with a unary modal operator $\langle a \rangle$, so put $\tau := (O, \rho)$ with $O := \{\langle a \rangle \mid a \in L\}$ and $\rho(\langle a \rangle) := 1$.

A nondeterministic τ -Kripke model is based on a labeled transition system $(S, (\rightarrow_a)_{a \in \mathbb{L}})$ which associates with each action a a binary relation $\rightarrow_a \subseteq S \times S$. Thus

$$s \models \langle a \rangle \varphi \Leftrightarrow \exists s' : s \rightarrow_a s' \wedge s' \models \varphi.$$

A stochastic τ -model is based on a labeled Markov transition system [18, 6, 9] $(S, (k_a)_{a \in \mathbb{L}})$ which associates with each action a a stochastic relation $k_a : S \rightsquigarrow S$. Thus

$$s \models \langle a \rangle_q \varphi \Leftrightarrow k_a(s)(\llbracket \varphi \rrbracket) \geq q,$$

and hence making a transition is replaced by a probability with which a transition can happen.

Variants of the logic $\mathfrak{M}\mathfrak{o}\mathfrak{d}_s(\tau, P)$ with $P = \emptyset$ were investigated in [18, 6] with a reference to the logic investigated by Hennessy and Milner; we refer to them also as *Hennessy–Milner logic*.

Example 5.3. The basic temporal language has two unary modal operators **F** (forward) and **B** (backward), so that $O = \{\mathbf{F}, \mathbf{B}\}$. A nondeterministic τ -Kripke model interprets the forward operator **F** through a relation $R \subseteq S \times S$ and the backward operator **B** through the converse R^\smile , so that

$$s \models \mathbf{B}\varphi \Leftrightarrow \exists t \in S : \langle t, s \rangle \in R \wedge t \models \varphi$$

holds.

A probabilistic interpretation interprets **F** through a stochastic relation $K : S \rightsquigarrow S$, so that

$$s \models \mathbf{F}_q \varphi \Leftrightarrow K(s)(\llbracket \varphi \rrbracket) \geq q.$$

The backward operator **B** is interpreted through the converse $K_\mu^\smile : S \rightsquigarrow S$, provided the state space S is Polish and an initial probability μ is given (the converse K_μ^\smile of K given μ is the stochastic relation $L : S \rightsquigarrow S$ such that

$$\int_S K(s)(\{s' \mid \langle s, s' \rangle \in B\}) \mu(ds) = \int_S L(s')(\{s \mid \langle s, s' \rangle \in B\}) \mu(ds')$$

holds for each Borel set $B \subseteq S \times S$; see [1] for algebraic and [10] for relational and measure-theoretic properties of the converse). Thus

$$s \models \mathbf{B}_q \varphi \Leftrightarrow K_\mu^\smile(s)(\llbracket \varphi \rrbracket) \geq q.$$

An easy calculation shows that

$$\begin{aligned} s \models \mathbf{B}_1 \mathbf{F}_1 \varphi &\Leftrightarrow K_\mu^\smile(s)(\{s' \mid K(s')(\llbracket \varphi \rrbracket) = 1\}) = 1 \\ &\Leftrightarrow \int_S K(s')(\llbracket \varphi \rrbracket) K_\mu^\smile(s)(ds') = 1. \end{aligned}$$

Note that the definition of the converse requires an initial probability (this is intuitively clear: if the probability for a backward running process is described, one has to say where to start). It is also noteworthy that a topological assumption has been made; if the state space is not a Polish space, then the technical arguments permitting the definition of the converse are not available.

Example 5.4. Arrow logic has three modal operators modelling reversal, composition, and skip, resp.; thus $O = \{\mathbf{1}, \otimes, \circ\}$ with respective arities $\rho(\mathbf{1}) = 0, \rho(\otimes) = 1, \rho(\circ) = 2$. The usual interpretation of arrow logic is done over a world of pairs, so the base state space is $S \times S$ for some S , with associated relations

$$\begin{aligned} R_{\mathbf{1}} &= \{\langle s, s \rangle \mid s \in S\}, \\ R_{\otimes} &= \{\langle \langle s_0, s_1 \rangle, \langle s_1, s_0 \rangle \rangle \mid s_0, s_1 \in S\}, \\ R_{\circ} &= \{\langle \langle s_0, s_1 \rangle, \langle s_0, s \rangle, \langle s, s_1 \rangle \rangle \mid s, s_0, s_1 \in S\} \end{aligned}$$

(see [3, Example 1.27]). Thus, e.g.,

$$\langle s, s' \rangle \models \phi \circ \psi \Leftrightarrow \exists s_0 : \langle s, s_0 \rangle \models \phi \wedge \langle s_0, s' \rangle \models \psi$$

and

$$\langle s, s' \rangle \models \otimes \phi \Leftrightarrow \langle s', s \rangle \models \phi.$$

Now assume that S is a Polish space and let $\mu \in \mathfrak{S}(S)$ be a subprobability. Put for $A \in \mathcal{B}(S \times S)$

$$\hat{\mu}(A) := \mu(\{s \in S \mid \langle s, s \rangle \in A\}).$$

Thus $\hat{\mu}$ transports a Borel set in S to a Borel set in the diagonal of $S \times S$. Denote by δ_a the Dirac measure on a .

Interpret the composition operator \circ_q through the stochastic relation

$$K_{\circ}(s, s') := \delta_s \otimes \hat{\mu} \otimes \delta_{s'}.$$

Note that the operator \otimes is somewhat overloaded: it denotes the modal operator for reversal, and the product operator for measures; the context should make it clear which version is meant.

We then obtain

$$\begin{aligned} K_{\circ}(s, s')(\llbracket \phi \rrbracket \times \llbracket \psi \rrbracket) &= (\delta_s \otimes \hat{\mu} \otimes \delta_{s'}) (\llbracket \phi \rrbracket \times \llbracket \psi \rrbracket) \\ &= \hat{\mu}(\{\langle s_1, s_2 \rangle \mid \langle s, s_1 \rangle \in \llbracket \phi \rrbracket, \langle s_2, s' \rangle \in \llbracket \psi \rrbracket\}) \\ &= \mu(\{s_1 \mid \langle s, s_1 \rangle \in \llbracket \phi \rrbracket, \langle s_1, s' \rangle \in \llbracket \psi \rrbracket\}). \end{aligned}$$

Consequently,

$$\langle s, s' \rangle \models \phi \circ_1 \psi \Leftrightarrow \langle s, s_1 \rangle \models \phi \wedge \langle s_1, s' \rangle \models \psi \text{ for } \mu\text{-almost all } s_1.$$

(Here μ -almost all s_1 means as usual that the set of all s_1 for which the property does not hold has μ -measure 0.) More generally,

$$\langle s, s' \rangle \models \phi \circ_q \psi \Leftrightarrow \langle s, s_1 \rangle \models \phi \wedge \langle s_1, s' \rangle \models \psi \quad \forall s_1 \text{ from a Borel set } S_0 \text{ with } \mu(S_0) \geq q.$$

Finally, put

$$K_{\otimes}(s, s') := \delta_{\langle s, s' \rangle};$$

then

$$\langle s, s' \rangle \models \otimes_q \phi \Leftrightarrow \langle s', s \rangle \models \phi$$

for all rational q with $0 \leq q \leq 1$ (which is independent of q). Let

$$K_{\mathbf{1}}(s, s') := \begin{cases} 0, & s \neq s', \\ \delta_{\langle s, s \rangle}, & s = s' \end{cases}$$

(here 0 is the null measure). Then

$$\langle s, s' \rangle \models \mathbf{1} \Leftrightarrow s = s'.$$

Note that in general we did exclude modal constants, i.e., modal operators of arity 0, when defining modal similarity types, since they will not contribute to the discussion. The example shows that it is possible to include them nevertheless.

The next example will deal with a logic that is used for model checking. Since this logic (sometimes called **pCTL***) incorporates both temporal and probabilistic properties, we leave the strict realm of modal logic.

The example requires some preparations. Fix a Polish state space S and assume that a family $(K_n)_{n \in \mathbb{N}}$ of probabilistic relations $K_n : S \rightsquigarrow S$ is given with $K_n(s)(S) = 1$ for each $s \in S, n \in \mathbb{N}$. $K_n(s)$ governs the state transitions at time n when the system is in state $s \in S$. We will have to deal with path probabilities, so let $S^\infty := \prod_{n \in \mathbb{N}} S$ be the set of all S -sequences, which is a Polish space in the product topology (this is the smallest topology on S^∞ which contains all sets of the form $\prod_{n \in \mathbb{N}} G_n$ where $G_n = S$ for all but a finite number of n , and all G_n are open in S). Similarly, the Borel sets $\mathcal{B}(S^\infty)$ are the smallest σ -algebra on S^∞ which contains all sets of the form $\prod_{n \in \mathbb{N}} B_n$ where $B_n = S$ for all but a finite number of n , and all B_n are Borel sets in S ; S^n denotes the n -fold product of S .

Define inductively a sequence $K^n : S \rightsquigarrow S^n$ by setting $K^1 := K_1$, and for $s \in S, A \subseteq S^{n+1}$ measurable

$$K^{n+1}(s)(A) := \int_{S^n} K_{n+1}(s_n)(\{s_n | \langle s_1, \dots, s_n \rangle \in A\}) K^n(s)(d\langle s_1, \dots, s_n \rangle).$$

Since K_n specifies probabilistically the n th state transition, $K^n(s)(A)$ gives the probability that the sequence $\langle s_1, \dots, s_n \rangle$ is an element of A , provided the system was initially in state s .

It is not difficult to see that the sequence $(K^n(s))_{n \in \mathbb{N}}$ forms an inverse system for each state $s \in S$, since for the measurable subset $A \subseteq S^n$ and for $s \in S$ the equality

$$K^{n+1}(s)(A \times X) = K^n(s)(A)$$

holds for each $n \in \mathbb{N}$. Denote for $s \in S$ by $K^\infty(s)$ the inverse limit of the inverse system $(K^n(s))_{n \in \mathbb{N}}$ of probabilities; see section 2.1. Then we claim that $K^\infty : S \rightsquigarrow S^\infty$ defines a stochastic relation.

For establishing this, we have to demonstrate that for each Borel subset $B \subseteq S^\infty$ the map $s \mapsto K^\infty(s)(B)$ is measurable. In fact, consider the set

$$\mathcal{S} := \{B \in \mathcal{B}(S^\infty) \mid s \mapsto K^\infty(s)(B) \text{ is measurable}\}.$$

Then \mathcal{S} is a σ -algebra which contains all sets of the form $\prod_{n \in \mathbb{N}} B_n$ where $B_n = S$ for all but a finite number of n , and all B_n are Borel sets in S . This is so since by construction

$$K^\infty(s) \left(\prod_{n \in \mathbb{N}} B_n \right) = K^m(s)(B_1 \times \dots \times B_m),$$

where m is an index such that $B_k = S$ for all $k > m$. Thus \mathcal{S} contains the generator for $\mathcal{B}(S^\infty)$, and since \mathcal{S} is comprised of Borel sets, \mathcal{S} must equal the Borel σ -algebra of S^∞ .

It may be noted that this representation is a bit simplistic in that a transition is assumed to be possible between two arbitrary states. Actually, one constructs an infinite tree from a relation modelling a more restricted transition graph. But this would have only increased the technical complexity without adding to the example.

Example 5.5. Model checking is a popular technique for verifying system properties. It is used in connection with stochastic techniques for the verification of non-functional properties in distributed systems; see, e.g., [4] where we show how the stochastic framework in the latter paper can be encoded in the approach proposed here. For a similar logic, see [2, 8]. An approach based on the mu-calculus is presented in [21] and could be dealt with in a similar way (this requires dealing appropriately with the \mathbb{E} -operator discussed in that paper).

A state formula ψ is given through the syntax

$$\psi ::= p \mid \top \mid \psi_1 \wedge \psi_2 \mid \neg\psi \mid [\varphi_1 \mathcal{U}_{\sim c} \varphi_2] \bowtie q,$$

where φ is given through

$$\varphi ::= p \mid \top \mid \varphi_1 \wedge \varphi_2$$

(here $p \in P, c \in \mathbb{N} \cup \{0\}, q \in \mathbb{Q} \cap [0, 1]$, and \sim, \bowtie are relational symbols from $\{<, >, \leq, \geq\}$).

A state formula $[\varphi_1 \mathcal{U}_{\sim c} \varphi_2] \bowtie q$ can only be used on the top level and cannot be nested. Various modal operators may be obtained as special cases, e.g., $\diamond_q \psi$ as $[\top \mathcal{U} \geq 0 \psi] \leq q$.

Validity for a state formula which does not contain the until operator is given in the usual way, and if σ is a path, then

$$\sigma \models [\varphi_1 \mathcal{U}_{\sim c} \varphi_2] \geq 1 \Leftrightarrow \exists x \sim c : (\sigma_x \models \varphi_2 \wedge \forall y \in [0, x[: \sigma_y \models \varphi_1)$$

(here σ_k is the k th component of σ).

For modelling validity of the \mathcal{U} -operator on paths, we assume that we are given a stochastic relation $K : S \rightsquigarrow S$. Construct from K the inverse system $K^n : S \rightsquigarrow S^n$, and let $K^\infty : S \rightsquigarrow S^\infty$ be the stochastic relation corresponding to the inverse limit of this system. Then we set

$$s \models [\varphi_1 \mathcal{U}_{\sim c} \varphi_2] \bowtie q \Leftrightarrow K^\infty(s) (\{\sigma \in S^\infty \mid \sigma^1 = s \wedge \sigma \models [\varphi_1 \mathcal{U}_{\sim c} \varphi_2] \geq 1\}) \bowtie q.$$

In this way satisfaction of state formulas through the stochastic system generated from the relation K can be modeled.

This example requires the construction of the inverse limit as a stochastic relation that relates states to infinite sequences of states which is in addition compatible with the transitions being done in each step.

5.2. Refinements. Given a nondeterministic and a stochastic interpretation, we want to compare both. Intuitively, the stochastic interpretation is more precise than its nondeterministic cousin, whereas nondeterministically we can only talk about possibilities (we can assign weights to these possibilities using probabilities). To say that after a certain input the output put will be a , b , or c conveys certainly less

information than saying that the probabilities for these outputs will be, resp., $p(a) = 1/100$, $p(b) = 1/50$, and $p(c) = 97/100$.

Since negation has its own problems, we will restrict ourselves to the negation free logic $\mathfrak{Mod}_1(\tau, P)$ and will deal with probabilistic relations (see section 2.2: the whole space is always assigned probability one).

DEFINITION 5.3. *Let \mathcal{R} and \mathcal{K} be a nondeterministic and a stochastic τ -Kripke model, resp., and assume that $K_\Delta(s)(S \times \cdots \times S^{\rho(\Delta)}) = 1$ holds for each $s \in S$ (we will call these models probabilistic). \mathcal{K} is said to refine \mathcal{R} ($\mathcal{K} \vdash \mathcal{R}$) iff*

$$\forall \varphi \in \mathfrak{Mod}_1(\tau, P) : \llbracket \varphi \rrbracket_{\mathcal{K}} \subseteq \llbracket \varphi \rrbracket_{\mathcal{R}}.$$

Consequently, given the interpretations \mathcal{K} and \mathcal{R} , we have $\mathcal{K} \vdash \mathcal{R}$ if $\mathcal{R}, s \models \varphi$ holds only if $\mathcal{K}, s \models \varphi$ is true for each formula φ in the negation free part of the logic.

We will investigate here the relationship between nondeterministic and stochastic satisfaction by showing that for each stochastic interpretation \mathcal{K} we can find a nondeterministic one \mathcal{R} with $\mathcal{K} \vdash \mathcal{R}$ by simply taking all possible state changes and making it into a Kripke model. Conversely, we will look into the possibility of refining a given nondeterministic Kripke model into a stochastic model. This requires some topological assumptions (for otherwise the notion of *all possible states* cannot be made precise). Thus from now on the state space S is a Polish space with its Borel sets as σ -algebra.

The set of all states possible for a probability μ on a Polish space X is captured through the support of a probability μ : Define $\text{supp}(\mu)$ as the smallest closed subset $F \subseteq X$ such that $\mu(F) = 1$; thus

$$\text{supp}(\mu) = \bigcap \{F \subseteq X \mid F \text{ is closed and } \mu(F) = 1\}.$$

It can be shown that $\mu(\text{supp}(\mu)) = 1$, and $x \in \text{supp}(\mu)$ iff $\mu(U) > 0$ for each neighborhood U of x ; this is exactly what we want.

PROPOSITION 5.4. *Let $\mathcal{K} = (S, (K_\Delta)_{\Delta \in O}, V)$ be a probabilistic τ -Kripke model. Define for the modal operator $\Delta \in O$ the set-valued map*

$$R_\Delta^{\mathcal{K}}(s) := \text{supp}(K_\Delta(s)).$$

Put

$$\mathcal{R}_{\mathcal{K}} := \left(S, (R_\Delta^{\mathcal{K}})_{\Delta \in O}, V \right);$$

then \mathcal{K} is a probabilistic refinement of $\mathcal{R}_{\mathcal{K}}$.

Proof. The proof proceeds by induction on the structure of the formulas. Assume that Δ is a modal operator and that we know that

$$\llbracket \varphi_i \rrbracket_{\mathcal{K}} \subseteq \llbracket \varphi_i \rrbracket_{\mathcal{R}_{\mathcal{K}}}$$

for $1 \leq i \leq \rho(\Delta)$. Now suppose

$$R_\Delta^{\mathcal{K}}, s \not\models \Delta_1(\varphi_1, \dots, \varphi_{\rho(\Delta)})$$

for some state s . Thus

$$R_\Delta^{\mathcal{K}}(s) \cap \llbracket \varphi_1 \rrbracket_{\mathcal{R}_{\mathcal{K}}} \times \cdots \times \llbracket \varphi_{\rho(\Delta)} \rrbracket_{\mathcal{R}_{\mathcal{K}}} = \emptyset,$$

and, consequently, by the hypothesis,

$$R_{\Delta}^{\mathcal{K}}(s) \cap \llbracket \varphi_1 \rrbracket_{\mathcal{R}} \times \cdots \times \llbracket \varphi_{\rho(\Delta)} \rrbracket_{\mathcal{R}} = \emptyset.$$

But this means

$$K_{\Delta}(s)(\llbracket \varphi_1 \rrbracket_{\mathcal{R}} \times \cdots \times \llbracket \varphi_{\rho(\Delta)} \rrbracket_{\mathcal{R}}) < 1;$$

hence

$$\mathcal{K}, s \not\models \Delta_1(\varphi_1, \dots, \varphi_{\rho(\Delta)}). \quad \square$$

Thus each probabilistic Kripke model carries a nondeterministic one with it, and it refines this companion (one is tempted to perceive this as a *nondeterministic shadow*: a shadow as a coarser, black-and-white image of a probably more colorful, picturesque, and graphic original).

It will be shown now that the converse of Proposition 5.4 is also true: Given a nondeterministic Kripke model, there exists a stochastic one refining it. Intuitively, and in the finite case, one simply assigns a uniform weight as a probability to all possible outcomes. This is basically what we will do here, too, but we have to be a bit more careful since in an uncountable setting this idea requires some additional underpinning.

It is immediate that the support yields a measurable relation for a probabilistic relation $K : Y \rightsquigarrow Z$: put

$$R_K := \{\langle y, z \rangle \in Y \times Z \mid z \in \text{supp}(K(y))\}.$$

Then

$$(\forall R_K)(F) = \{y \in Y \mid K(y)(F) = 1\}$$

is true for the closed set $F \subseteq Z$, and

$$(\exists R_K)(G) = \{y \in Y \mid K(y)(G) > 0\}$$

holds for the open set $G \subseteq Z$. Both sets are measurable.

It is also plain that a representation of R through a stochastic relation K which is given by

$$(*) \forall y \in Y : R(y) = \text{supp}(K(y))$$

implies that R has to be a measurable relation.

Given a set-valued relation R , a probabilistic relation K with $(*)$ can be found. For this, R has to take closed values, and a condition of measurability is imposed. We obtain from [10] the following existential statement (which depends on the existence of a sufficient number of measurable selectors for a measurable relation; see Proposition 2.1).

LEMMA 5.5. *Let $R \subseteq Y \times Z$ be a measurable relation for Z Polish. There exists a probabilistic relation $K : Y \rightsquigarrow Z$ such that $R(y) = \text{supp}(K(y))$ holds for each $y \in Y$.*

Thus we can find a probabilistic Kripke structure refining a given nondeterministic one, provided we impose a measurability condition.

PROPOSITION 5.6. *Suppose $\mathcal{R} := (S, (R_{\Delta})_{\Delta \in O}, V)$ is a nondeterministic τ -Kripke model such that*

1. $V(p) \in \mathcal{B}(S)$ for all $p \in P$,
2. R_Δ is a measurable relation on $S \times S^{\rho(\Delta)}$ for each $\Delta \in O$.

Then there exists a probabilistic τ -Kripke model $\mathcal{K} = (S, (K_\Delta)_{\Delta \in O}, V)$ with $\mathcal{K} \vdash \mathcal{R}$.

Proof. Applying Lemma 5.5, find for each modal operator $\Delta \in O$ a transition probability $K_\Delta : S \rightsquigarrow S^{\rho(\Delta)}$ such that

$$\forall s \in S : R_\Delta(s) = \text{supp}(K_\Delta(s))$$

holds. The argumentation in the proof of Lemma 5.4 establishes the claim. \square

It is clear that the probabilistic τ -Kripke model is underspecified by merely requiring to be a refinement to a nondeterministic one. This is supported through the following corollary.

COROLLARY 5.7. *Let \mathcal{R} be a nondeterministic τ -Kripke model satisfying the conditions of Proposition 5.6. Assume that $\mathcal{K}_i = (S, (K_{\Delta,i})_{\Delta \in O}, V)$ is a probabilistic τ -Kripke model with $\mathcal{K}_i \vdash \mathcal{R}$ for each $i \in \mathbb{N}$. Let $(\alpha_i)_{i \in \mathbb{N}}$ be a sequence of positive real numbers such that $\sum_{i \in \mathbb{N}} \alpha_i = 1$, and define for $\Delta \in O$ the stochastic relation*

$$K_\Delta(s) := \sum_{i \in \mathbb{N}} \alpha_i \cdot K_{\Delta,i}(s).$$

Then

$$(S, (K_\Delta)_{\Delta \in O}, V) \vdash \mathcal{R}.$$

Proof. 1. Let $(\mu_i)_{i \in \mathbb{N}}$ be a sequence of probability measures. Since all α_i are positive, the definition of the support function yields that

$$\text{supp} \left(\sum_{i \in \mathbb{N}} \alpha_i \cdot \mu_i \right) = \left(\bigcup_{i \in \mathbb{N}} \text{supp}(\mu_i) \right)^{\text{cl}}$$

holds, $(\cdot)^{\text{cl}}$ denoting topological closure. Thus R_Δ equals $\text{supp}(K_\Delta)$.

2. The assertion now follows from Proposition 5.6. \square

Thus we know not only that a probabilistic τ -Kripke model is the refinement of a probabilistic one, but also that refinements offer a considerable degree of freedom, because they are closed under countable convex combinations (in fact, it can also be shown that it is closed under integration as the generalization of convex combinations). This supports the intuitive feeling that a probabilistic model conveys much more information than a nondeterministic one, but that it is also much harder to obtain.

6. Bisimulations for Kripke models. This section investigates morphisms for stochastic τ -Kripke models; we want to know whether bisimilarity and mutually identical theories are equivalent also for this general case. To this end we first discuss morphisms that are based on morphisms for stochastic relations (after all, a τ -Kripke model contains a family of stochastic relations), indicate that this notion of morphism is not adequate for our purposes, and propose the notion of a strong morphism. We show that strong morphisms are suitable for our purposes.

Fix the modal similarity type $\tau = (O, \rho)$ again. Assume first that the set P of propositional letters is empty, rendering the initial discussion a bit less technical. Then a stochastic τ -Kripke model $\mathcal{K} := (S, (K_\Delta)_{\Delta \in O})$ is determined through the

Polish state space S and the family $K_\Delta : S \rightsquigarrow S^{\rho(\Delta)}$ of stochastic relations. A morphism

$$\Phi : (S, (K_\Delta)_{\Delta \in O}) \rightarrow (S', (K'_\Delta)_{\Delta \in O})$$

for stochastic τ -Kripke models is then a family

$$\Phi = ((\phi_\Delta, \psi_\Delta)_{\Delta \in O})$$

of morphisms

$$(\phi_\Delta, \psi_\Delta) : (S, S^{\rho(\Delta)}, K_\Delta) \rightarrow (S', (S')^{\rho(\Delta)}, K'_\Delta)$$

for the associated relations.

Consider a modal operator Δ . The σ -algebra \mathcal{A}_Δ generated by

$$\{[\![\varphi_1]\!]_{\mathcal{K}} \times \cdots \times [\![\varphi_{\rho(\Delta)}]\!]_{\mathcal{K}} \mid \varphi_1, \dots, \varphi_{\rho(\Delta)} \in \mathfrak{Mod}_s(\tau, P)\}$$

is evidently countably generated, thus giving rise to a smooth equivalence relation β_Δ on $S^{\rho(\Delta)}$, and the relation

$$s\alpha_\Delta s' \Leftrightarrow \forall B \in \mathcal{A}_\Delta : K_\Delta(s)(B) = K_\Delta(s')(B)$$

is smooth due to \mathcal{A}_Δ being countably generated. Consequently, $(\alpha_\Delta, \beta_\Delta)$ is a congruence for $K_\Delta : S \rightsquigarrow S^{\rho(\Delta)}$, and if \mathcal{K} is nondegenerate, this congruence is nontrivial.

Let $\mathcal{K}' = (S', (K'_\Delta)_{\Delta \in O})$ be another τ -Kripke model which is equivalent to the first one in the sense that for the states the corresponding theories mutually coincide; the following definition will be more precise.

DEFINITION 6.1. *The stochastic τ -Kripke models \mathcal{K} and \mathcal{K}' are said to be equivalent ($\mathcal{K} \sim \mathcal{K}'$) iff given $s \in S$ there exists $s' \in S'$ such that $Th_{\mathcal{K}}(s) = Th_{\mathcal{K}'}(s')$ and vice versa.*

Assume both \mathcal{K} and \mathcal{K}' are nondegenerate. Construct for \mathcal{K}' the congruence $(\alpha'_\Delta, \beta'_\Delta)$ for each modal operator Δ as above; then it can be shown that $\mathcal{K} \sim \mathcal{K}'$ implies that the congruences $(\alpha_\Delta, \beta_\Delta)$ and $(\alpha'_\Delta, \beta'_\Delta)$ are equivalent. From Proposition 4.5 we see that K_Δ and K'_Δ are bisimilar for each modal operator Δ , so that there exists a span of morphisms

$$(S, S^{\rho(\Delta)}, K_\Delta) \xleftarrow{(\phi_\Delta, \psi_\Delta)} (A_\Delta, B_\Delta, M_\Delta) \xrightarrow{(\phi'_\Delta, \psi'_\Delta)} (S', (S')^{\rho(\Delta)}, K'_\Delta).$$

This is rather satisfying from the point of view of stochastic relations, but not when considering stochastic τ -Kripke models. This is so since in general $((A_\Delta, B_\Delta, M_\Delta)_{\Delta \in O})$ fails to be such a model because there is no way to guarantee that all A_Δ coincide with, say, a Polish space T , and so that B_Δ equals $T^{\rho(\Delta)}$.

Consequently, we have to strengthen the requirements for a morphism in order to achieve some uniformity. This will be done now, and we admit propositional letters again.

The basic idea is to have just one map ϕ between the state spaces so that

$$K'_\Delta(\phi(s))(A) = K_\Delta(s)(\{\langle s_1, \dots, s_{\rho(\Delta)} \rangle \mid \langle \phi(s_1), \dots, \phi(s_{\rho(\Delta)}) \rangle \in A\})$$

holds for each state $s \in S$ and each Borel set $A \subseteq (S')^{\rho(\Delta)}$, making the diagram

$$\begin{array}{ccc} S & \xrightarrow{\phi} & S' \\ K_{\Delta} \downarrow & & \downarrow K'_{\Delta} \\ \mathfrak{S}(S^{\rho(\Delta)}) & \xrightarrow{\mathfrak{S}(\phi^{\rho(\Delta)})} & \mathfrak{S}((S')^{\rho(\Delta)}) \end{array}$$

commutative (where $\phi^n : \langle x_1, \dots, x_n \rangle \mapsto \langle \phi(x_1), \dots, \phi(x_n) \rangle$ distributes ϕ into the components), and we want to have $s \in V(p)$ iff $\phi(s) \in V'(p)$ for each propositional letter. This leads to the following definition.

DEFINITION 6.2. Let $\mathcal{K} := (S, (K_{\Delta})_{\Delta \in O}, V)$ and $\mathcal{K}' := (S', (K'_{\Delta})_{\Delta \in O}, V')$ be stochastic τ -Kripke models. A strong morphism $\phi : \mathcal{K} \rightarrow \mathcal{K}'$ is determined through a measurable and surjective map $\phi : S \rightarrow S'$ so that these conditions are satisfied:

1. $\forall p \in P : V(p) = \phi^{-1}[V'(p)]$,
2. for each modal operator Δ ,

$$K'_{\Delta} \circ \phi = \mathfrak{S}(\phi^{\rho(\Delta)}) \circ K_{\Delta}$$

holds.

Thus, if $\phi : \mathcal{K} \rightarrow \mathcal{K}'$ is a strong morphism, then

$$(\phi, \phi^{\rho(\Delta)}) : (S, S^{\rho(\Delta)}, K_{\Delta}) \rightarrow (S', (S')^{\rho(\Delta)}, K'_{\Delta})$$

is a morphism between the corresponding stochastic relations for each modal operator $\Delta \in O$. Note that we also take the propositional letters into account.

It is clear that stochastic τ -Kripke models over general measurable spaces form a category **pKripke** with this notion of morphism, because the composition of strong morphisms is again a strong morphism, and because the identity is a strong morphism, too. Furthermore, each modal operator Δ induces a functor $F_{\Delta} : \mathbf{pKripke} \rightarrow \mathbf{Stoch}$ which forgets all but K_{Δ} . We will below make (rather informal) use of this functor.

Because we work on the safe grounds of a category, we have bisimulations at our disposal, which can be defined again as spans of strong morphisms.

DEFINITION 6.3. The stochastic τ -Kripke models \mathcal{K}_1 and \mathcal{K}_2 are called strongly bisimilar iff

1. there exist a mediating stochastic τ -Kripke model \mathcal{M} and strong morphisms

$$\mathcal{K}_1 \xleftarrow{\phi_1} \mathcal{M} \xrightarrow{\phi_2} \mathcal{K}_2,$$

2. the σ -algebra

$$\phi_1^{-1}[\mathcal{B}(S_1)] \cap \phi_2^{-1}[\mathcal{B}(S_2)]$$

is nontrivial (here S_i is the state space of \mathcal{K}_i , $i = 1, 2$).

Since the product σ -algebra is the smallest σ -algebra which contains all the measurable rectangles, it is not difficult to see that

$$\phi_1^{-1}[\mathcal{B}(S_1)] \cap \phi_2^{-1}[\mathcal{B}(S_2)]$$

is nontrivial iff for each modal operator $\Delta \in O$ the σ -algebra

$$\bigotimes_{i=1}^{\rho(\Delta)} \phi_1^{-1} [\mathcal{B}(S_1)] \cap \bigotimes_{i=1}^{\rho(\Delta)} \phi_2^{-1} [\mathcal{B}(S_2)]$$

is nontrivial. Thus condition 2 in Definition 6.3 will imply that this notion of bisimilarity is compatible to the one used for stochastic relations in general.

We will show that $\mathcal{K} \sim \mathcal{K}'$ iff \mathcal{K} and \mathcal{K}' are strongly bisimilar, provided the models are based on Polish spaces. Fix the stochastic τ -Kripke models $\mathcal{K} := (S, (K_\Delta)_{\Delta \in O}, V)$ and $\mathcal{K}' := (S', (K'_\Delta)_{\Delta \in O}, V')$.

It is well known that morphisms preserve theories for the Hennessy–Milner logic [6]. This is also true for stochastic relations.

LEMMA 6.4. *If $\phi : \mathcal{K} \rightarrow \mathcal{K}'$ is a strong morphism, then*

$$Th_{\mathcal{K}}(s) = Th_{\mathcal{K}'}(\phi(s))$$

holds for all states $s \in S$.

Proof. 1. We show by induction on the formula $\varphi \in \mathfrak{Mod}_s(\tau, P)$ that

$$\mathcal{K}, s \models \varphi \Leftrightarrow \mathcal{K}', \phi(s) \models \varphi$$

holds; putting it slightly differently, we want to show that

$$(*) \llbracket \varphi \rrbracket_{\mathcal{K}} = \llbracket \varphi \rrbracket_{\mathcal{K}'}$$

for all these φ .

2. If $\varphi = p \in P$, this follows from $V(p) = \phi^{-1}[V'(p)]$. The interesting case in the induction step is the application of an n -ary modal operator Δ_q with rational q . Suppose the assertion is true for $\llbracket \varphi_1 \rrbracket_{\mathcal{K}}, \dots, \llbracket \varphi_n \rrbracket_{\mathcal{K}}$; then

$$\begin{aligned} \mathcal{K}, s \models \Delta_q(\varphi_1, \dots, \varphi_n) &\Leftrightarrow K_\Delta(s)(\llbracket \varphi_1 \rrbracket_{\mathcal{K}} \times \dots \times \llbracket \varphi_n \rrbracket_{\mathcal{K}}) \geq q \\ &\Leftrightarrow K_\Delta(s)((\phi^n)^{-1}[\llbracket \varphi_1 \rrbracket_{\mathcal{K}'} \times \dots \times \llbracket \varphi_n \rrbracket_{\mathcal{K}'}]) \geq q \quad (\dagger) \\ &\Leftrightarrow (\mathfrak{S}(\phi^n) \circ K_\Delta)(s)(\llbracket \varphi_1 \rrbracket_{\mathcal{K}'} \times \dots \times \llbracket \varphi_n \rrbracket_{\mathcal{K}'})) \geq q \\ &\Leftrightarrow K'_\Delta(\phi(s))(\llbracket \varphi_1 \rrbracket_{\mathcal{K}'} \times \dots \times \llbracket \varphi_n \rrbracket_{\mathcal{K}'}) \geq q \quad (\ddagger) \\ &\Leftrightarrow \mathcal{K}', \phi(s) \models \Delta_q(\varphi_1, \dots, \varphi_n). \end{aligned}$$

In (\dagger) we use reformulation $(*)$ for the induction hypothesis, and in (\ddagger) we make use of the defining equation of a (strong) morphism. \square

Define the equivalence relation α on state space S through

$$s_1 \alpha s_2 \Leftrightarrow Th_{\mathcal{K}}(s_1) = Th_{\mathcal{K}}(s_2);$$

thus two states are α -equivalent iff they satisfy exactly the same formulas in $\mathfrak{Mod}_s(\tau, P)$; in a similar way α' is defined on S' . Because we have at most countably many formulas, α and α' are smooth equivalence relations. Define the equivalence relation β_Δ on $S^{\rho(\Delta)}$ through

$$\langle s_1, \dots, s_{\rho(\Delta)} \rangle \beta_\Delta \langle t_1, \dots, t_{\rho(\Delta)} \rangle \Leftrightarrow s_1 \alpha t_1 \wedge \dots \wedge s_{\rho(\Delta)} \alpha t_{\rho(\Delta)};$$

then β_Δ is smooth, and we know that the σ -algebra of β -invariant sets can be written in terms of the α -invariant sets, viz., $\mathcal{INV}(\mathcal{B}(S^{\rho(\Delta)}), \beta_\Delta) = \bigotimes_{i=1}^{\rho(\Delta)} \mathcal{INV}(\mathcal{B}(S), \alpha)$ (see Lemma 3.3). The relation β'_Δ is defined in the same way for α' .

The equivalence of \mathcal{K} and \mathcal{K}' makes these relations into equivalent congruences.

LEMMA 6.5. *If $\mathcal{K} \sim \mathcal{K}'$ for the nondegenerate Kripke models \mathcal{K} and \mathcal{K}' , then (α, β_Δ) and (α', β'_Δ) are equivalent and nontrivial congruences on the stochastic relations $F_\Delta(\mathcal{K})$ and $F_\Delta(\mathcal{K}')$.*

Proof. 1. The equivalence relations involved are all smooth, so it first has to be demonstrated that each pair indeed forms a congruence. Assume that $s_1 \alpha s_2$ holds; then

$$K_\Delta(s_1)(\llbracket \varphi_1 \rrbracket_{\mathcal{K}} \times \cdots \times \llbracket \varphi_{\rho(\Delta)} \rrbracket_{\mathcal{K}}) = K_\Delta(s_2)(\llbracket \varphi_1 \rrbracket_{\mathcal{K}} \times \cdots \times \llbracket \varphi_{\rho(\Delta)} \rrbracket_{\mathcal{K}})$$

follows (otherwise we could find a rational number q with $\mathcal{K}, s_1 \models \Delta_q(\varphi_1, \dots, \varphi_{\rho(\Delta)})$ but $\mathcal{K}, s_2 \not\models \Delta_q(\varphi_1, \dots, \varphi_{\rho(\Delta)})$ or vice versa). Because

$$\mathcal{B}_0 := \{\llbracket \varphi_1 \rrbracket_{\mathcal{K}} \times \cdots \times \llbracket \varphi_{\rho(\Delta)} \rrbracket_{\mathcal{K}} \mid \varphi_1, \dots, \varphi_{\rho(\Delta)} \in \mathfrak{Mod}_s(\tau, P)\}$$

forms a generator for $\mathcal{INV}(\mathcal{B}(S^{\rho(\Delta)}), \beta_\Delta)$, we see that (α, β_Δ) is a congruence for $F_\Delta(\mathcal{K})$. The same arguments show that also (α', β'_Δ) is a congruence for $F_\Delta(\mathcal{K}')$.

2. $\mathcal{A}_0 := \{\llbracket \varphi \rrbracket_{\mathcal{K}} \mid \varphi \in \mathfrak{Mod}_s(\tau, P)\}$ is a countable generator of the σ -algebra $\mathcal{INV}(\mathcal{B}(S), \alpha)$, and since the logic is closed under conjunction, \mathcal{A}_0 is closed under finite intersections. Given $s \in S$ there exists $s' \in S'$ such that $Th_{\mathcal{K}}(s) = Th_{\mathcal{K}'}(s')$ holds; define $\Upsilon([s]_\alpha) := [s']_{\alpha'}$, and then $\Upsilon : S/\alpha \rightarrow S'/\alpha'$ is well defined, and $\Upsilon_{\llbracket \varphi \rrbracket_{\mathcal{K}}} = \llbracket \varphi \rrbracket_{\mathcal{K}'}$ holds. Consequently, $\{\Upsilon_A \mid A \in \mathcal{A}_0\}$ generates $\mathcal{INV}(\mathcal{B}(S'), \alpha')$, and the construction implies that

$$\bigcap \{\Upsilon_A \mid s \in A \in \mathcal{A}_0\} \cap \bigcap \{S' \setminus \Upsilon_A \mid s \notin A \in \mathcal{A}_0\} = [s']_{\alpha'}.$$

Hence α spawns α' via $(\Upsilon, \mathcal{A}_0)$.

3. The construction of β_Δ implies that

$$\langle [s_1, \dots, s_{\rho(\Delta)}] \rangle_{\beta_\Delta} = [s_1]_\alpha \times \cdots \times [s_{\rho(\Delta)}]_\alpha$$

holds. An argument very similar to that used above shows that β_Δ spawns β'_Δ via (Θ, \mathcal{B}_0) , where

$$\Theta : \langle [s_1, \dots, s_{\rho(\Delta)}] \rangle_{\beta_\Delta} \mapsto \Upsilon([s_1]_\alpha) \times \cdots \times \Upsilon([s_{\rho(\Delta)}]_\alpha),$$

and \mathcal{B}_0 is defined above.

4. An argumentation very close to the first part of the proof shows that $Th_{\mathcal{K}}(s) = Th_{\mathcal{K}'}(s')$ for $s \in S, s' \in S'$ implies for all formulas $\varphi_1, \dots, \varphi_{\rho(\Delta)}$ that

$$K_\Delta(s)(\llbracket \varphi_1 \rrbracket_{\mathcal{K}} \times \cdots \times \llbracket \varphi_{\rho(\Delta)} \rrbracket_{\mathcal{K}}) = K'_\Delta(s')(\llbracket \varphi_1 \rrbracket_{\mathcal{K}'} \times \cdots \times \llbracket \varphi_{\rho(\Delta)} \rrbracket_{\mathcal{K}'})$$

(see part 2 of the proof of Lemma 6.4). Thus $(\alpha, \beta_\Delta) \times (\alpha', \beta'_\Delta)$, and in the same way, interchanging the roles of \mathcal{K} and \mathcal{K}' , we infer $(\alpha', \beta'_\Delta) \times (\alpha, \beta_\Delta)$.

5. Because \mathcal{K} is nondegenerate, the σ -algebra

$$\mathcal{INV}(\mathcal{B}(S), \alpha) = \sigma(\{\llbracket \varphi \rrbracket_{\mathcal{K}} \mid \varphi \in \mathfrak{Mod}_s(\tau, P)\})$$

is nontrivial. Because

$$\mathcal{INV}(\mathcal{B}(S^{\rho(\Delta)}), \beta_\Delta) = \bigotimes_{i=1}^{\rho(\Delta)} \mathcal{INV}(\mathcal{B}(S), \alpha),$$

we see that $\mathcal{I}NV(\mathcal{B}(S^{\rho(\Delta)}), \beta_\Delta)$ contains a set of the form $B^{\rho(\Delta)}$ for some B with $\emptyset \neq B \neq S$. Thus we may conclude that β_Δ is not the universal relation. Thus (α, β_Δ) is a nontrivial congruence. Replacing \mathcal{K} by \mathcal{K}' , this is also established for the congruence (α', β'_Δ) . This completes the proof. \square

Accordingly, we know from Proposition 4.5 that for equivalent Kripke models \mathcal{K} and \mathcal{K}' and for each modal operator Δ the stochastic relations $F_\Delta(\mathcal{K})$ and $F_\Delta(\mathcal{K}')$ are bisimilar. All the mediating relations can be collected to form a mediating Kripke model. This requires, however, that we have introductory knowledge of the internal structure of the semipullback which is constructed along the way, as we will see now in the proof of the following main result, the Hennessy–Milner theorem for stochastic τ -Kripke models.

THEOREM 6.6. *Assume that \mathcal{K} and \mathcal{K}' are nondegenerate stochastic τ -Kripke models over Polish spaces; then the following statements are equivalent:*

1. \mathcal{K} and \mathcal{K}' are strongly bisimilar,
2. $\mathcal{K} \sim \mathcal{K}'$.

Proof. 1. Since “1 \Rightarrow 2” follows from Lemma 6.4, we may concentrate on the proof for “2 \Rightarrow 1.”

2. Since $\mathcal{K} \sim \mathcal{K}'$, we know from Lemma 6.5 that the congruences (α, β_Δ) and (α', β'_Δ) are equivalent for each modal operator Δ . Let $\mathcal{M}_\Delta = (M_\Delta, N_\Delta, L_\Delta)$ be the mediating stochastic relation, which exists by Proposition 4.5. Theorem 2.2 shows that ($n := \rho(\Delta)$)

$$\begin{aligned} M_\Delta &= \{ \langle s, s' \rangle \in S \times S' \mid \langle s, \alpha + \alpha' \rangle s' \}, \\ N_\Delta &= \{ \langle s_1, s'_1, \dots, s_n, s'_n \rangle \in (S \times S')^n \mid s_i (\alpha + \alpha') s'_i \text{ for } 1 \leq i \leq n \}. \end{aligned}$$

These may be made into Polish spaces. Note that $S'' := M_\Delta$ does not depend at all on the modal operator, and that N_Δ depends only on its arity. Furthermore, we may infer for the **P** – **Stoch**-morphisms

$$F_\Delta(\mathcal{K}) \xleftarrow{f_\Delta} \mathcal{M}_\Delta \xrightarrow{f'_\Delta} F_\Delta(\mathcal{K}')$$

that

$$f_\Delta = (\pi_{1,S}, \pi_{1,S}^n), f'_\Delta = (\pi_{2,S'}, \pi_{2,S'}^n)$$

holds, where the π denote the projections. Now define for the propositional letter $p \in P$

$$W(p) := \{ \langle s, s' \rangle \in M_\Delta \mid s \in V(p), s' \in V'(p) \}.$$

Then it is immediate that the equations $W(p) = \pi_1^{-1}[V(p)] = \pi_2^{-1}[V'(p)]$ hold. Consequently, $\mathcal{M} := (S'', (L_\Delta)_{\Delta \in \mathcal{O}}, W)$ is a stochastic τ -Kripke model with

$$\mathcal{K} \xleftarrow{\pi_{1,S}} \mathcal{M} \xrightarrow{\pi_{2,S'}} \mathcal{K}'$$

in **pKripke**.

3. Finally, we need to show that the σ -algebra $\pi_{1,S}^{-1}[\mathcal{B}(S)] \cap \pi_{2,S'}^{-1}[\mathcal{B}(S')]$ is non-trivial. This is essentially the same argument as the one used in the third part of the proof of Proposition 4.5. Since \mathcal{K} is nondegenerate, we can find a formula φ with $\emptyset \neq \llbracket \varphi \rrbracket_{\mathcal{K}} \neq S$. The set $\llbracket \varphi \rrbracket_{\mathcal{K}}$ is an α -invariant Borel subset of S . We know from

the proof of Lemma 6.5 that α spawns α' via $(\Upsilon, \{\llbracket \varphi \rrbracket_{\mathcal{K}} \mid \phi \in \mathfrak{Mod}_{\mathfrak{s}}(\tau, P)\})$ for some suitably chosen Υ . Thus

$$\pi_{1,S}^{-1} [\llbracket \varphi \rrbracket_{\mathcal{K}}] = \pi_{2,S'}^{-1} [\Upsilon_{\llbracket \varphi \rrbracket_{\mathcal{K}}}] .$$

Consequently, we see that

$$\pi_{1,S}^{-1} [\llbracket \varphi \rrbracket_{\mathcal{K}}] \in \pi_{1,S}^{-1} [\mathcal{B}(S)] \cap \pi_{2,S'}^{-1} [\mathcal{B}(S')] .$$

Since $\emptyset \neq \llbracket \varphi \rrbracket_{\mathcal{K}} \neq S$ we conclude that $\emptyset \neq \pi_{1,S}^{-1} [\llbracket \varphi \rrbracket_{\mathcal{K}}] \neq M_{\Delta}$, and hence the σ -algebra in question is indeed not trivial. \square

Looking back at the development, it may be noted that Theorem 6.6 is derived from Proposition 4.5, hence from a condition that arose from the consideration of stochastic relations alone. This is in marked contrast to the proofs proposed in [6, 9] which start from the logic and develop the properties of equivalent congruences implicitly.

7. Related work. The investigation of congruences and their relationship to bisimilarity is new. Factoring stochastic relations has been introduced and studied in [11]. Special cases were considered in [6] and in [9] mainly in the realm of labeled Markov transition systems.

The research reported in [6] takes as a basic scenario an analytic state space and considers universally measurable transition functions. A Hennessy–Milner theorem is proved; the proof’s idea is to produce a cospan of morphisms through injections into a suitably factored sum. This idea has left its traces in various parts of the present paper. But the situation considered here is structurally subtly different: universal measurability, as assumed in [6], requires a somewhat elaborate completion process using all finite measures on that space. The present paper requires as a measurable structure merely the Borel sets of an analytic space. They are structurally much simpler and do not need additional considerations, since they are given through the morphisms of measurable spaces and nothing else (so one could work with them even if one would want to do without the real numbers). This more general approach has become possible through the observation that semipullbacks exist on analytic spaces with their Borel sets (a result that was not yet available for [6]). Desharnais and Panangaden [8] use an interplay between bisimulations (which are called here congruences) for showing that the logic *continuous time stochastic logic* which is discussed in [2] for model checking has also a property similar to the one discussed in Theorem 6.6: two states in a Markov process for the language satisfy the same formula from a given closed set F of formulas iff they are F -bisimilar [8, Theorem 6.3]. This is proved directly, without the benefit of a general criterion for bisimilarity.

In [9] a generalization of [6] is established for those labeled Markov transition systems which work over a Polish (rather than an analytic) state space and which have a certain smallness property. Technically, this property makes sure that the factor space is well behaved again. This technical condition is lifted in the present paper. This is so since general analytic spaces are considered, and the technique of factoring stochastic relations is better understood. Apart from a much wider class of modal logics which can be dealt with now (as witnessed in section 5.1), the present paper proposes a more general technical approach.

A stochastic relation $S \rightsquigarrow S$ may be considered as a coalgebra for the subprobability functor \mathfrak{S} on the category of analytic spaces; hence there are some ties to coalgebras. In [20] the probability functor \mathfrak{P} is considered on measurable spaces

that are endowed with the initial σ -algebra for the evaluation mapping $\mu \mapsto \mu(E)$. Given a discrete space I , final coalgebras for a functor derived from \mathfrak{P} yielding a type space over category \mathbf{Meas}^I are discussed, where \mathbf{Meas} is the category of measurable spaces with measurable maps as morphisms. Besides establishing the existence of a final coalgebra for the functor through satisfied theories, the main result states that functors polynomial in the original type functor have final coalgebras, too. In [5] coalgebraic simulation is considered, and one of the application areas for the discussion is probabilistic transition systems. They are modeled as coalgebras for the functor that assigns each set its discrete probability measures.

8. Conclusion and further work. The main technical result of this paper is the characterization of bisimilarity through suitable congruences, and through isomorphic factors. It bears fruit in establishing a Hennessy–Milner theorem for stochastic interpretations of modal logic. This paper proposes these contributions.

1. The notion of equivalent congruences is introduced and studied. Stochastic relations having equivalent congruences are shown to be bisimilar. As a corollary it is shown that isomorphic factors imply bisimilarity, too. It is shown that bisimilar relations also have isomorphic factor spaces, but this depends on a rather strong topological condition. We conjecture that the existence of isomorphic factor spaces and bisimilarity are equivalent for generic analytic spaces.

2. Stochastic Kripke models are introduced as a generalization of the well-known labeled Markov transition systems for general modal logics. A refinement relation between these models and their nondeterministic counterparts is investigated.

3. For stochastic Kripke models we propose the notion of a strong morphism, and, correspondingly, of strong bisimulations. A stochastic version of the Hennessy–Milner theorem of the equivalence of bisimilarity and mutually identical theories is established.

Examples show how some popular logics are interpreted through a stochastic Kripke model, and this opens the avenue for further research. It should be interesting to see how other modal logics are interpreted. A prime candidate is propositional dynamic logic (PDL) with its rich interaction among the modal operators that would have to be reflected in a suitable structure for the stochastic relations interpreting it. Investigating the interrelationship among a probabilistic interpretation, Kozen’s semantics of probabilistic programs [16], and probabilistic predicate transformers [19] is expected to give new insights into PDL as well as probabilistic program semantics and, incidentally, the algebraic properties of stochastic relations.

As the work in [2, 8] witnesses, that there are interesting links among stochastic relations, Markov processes, and model checking. It will be most helpful for model checking, and for a better understanding of probabilistic processes, to study this link closely.

Acknowledgments. The author appreciates and wants to thank the referees for providing insightful comments and suggestions which helped improve the paper.

REFERENCES

- [1] S. ABRAMSKY, R. BLUTE, AND P. PANANGADEN, *Nuclear and trace ideal in tensored *-categories*, J. Pure Appl. Algebra, 143 (1999), pp. 3–47.
- [2] C. BAIER, B. HAVERKORT, H. HERMANN, AND J.-P. KOERT, *Model-checking algorithms for continuous time Markov chains*, IEEE Trans. Softw. Eng., 29 (2003), pp. 524–541.
- [3] P. BLACKBURN, M. DE RJIKE, AND Y. VENEMA, *Modal Logic*, Cambridge Tracts Theoret. Comput. Sci. 53, Cambridge University Press, Cambridge, UK, 2001.

- [4] J. BRYANS, H. BOWMAN, AND J. DERRICK, *Model checking stochastic automata*, ACM Trans. Comput. Log., 4 (2003), pp. 452–492.
- [5] C. CÎRSTEA, *On logics for coalgebraic simulation*, in Proceedings of the 7th International Workshop on Coalgebraic Methods in Computer Science (Barcelona, Spain), Electron Notes Theor. Comput. Sci. 106, Elsevier, Amsterdam, 2004, pp. 63–90.
- [6] J. DESHARNAIS, A. EDALAT, AND P. PANANGADEN, *Bisimulation for labelled Markov processes*, Inform. and Comput., 179 (2002), pp. 163–193.
- [7] J. DESHARNAIS, R. JAGADEESAN, V. GUPTA, AND P. PANANGADEN, *Approximating labeled Markov processes*, in Proceedings of the 15th Annual IEEE Symposium on Logic in Computer Science, Santa Barbara, CA, 2000, IEEE Computer Society, Los Alamitos, CA, 2000, pp. 95–106.
- [8] J. DESHARNAIS AND P. PANANGADEN, *Continuous stochastic logic characterizes bisimulation of continuous-time Markov processes*, J. Log. Algebr. Program., 56 (2003), pp. 99–115.
- [9] E.-E. DOBERKAT, *Semipullbacks and bisimulations in categories of stochastic relations*, in Proceedings of the ICALP '03, Lecture Notes in Comput. Sci. 2719, Springer-Verlag, Berlin, 2003, pp. 996–1007.
- [10] E.-E. DOBERKAT, *The converse of a stochastic relation*, J. Log. Algebr. Program., 62 (2005), pp. 133–154.
- [11] E.-E. DOBERKAT, *Factoring stochastic relations*, Inform. Process. Lett., 90 (2004), pp. 161–166.
- [12] E.-E. DOBERKAT, *Look: Simple stochastic relations are just, well, simple*, in Proceedings of the Conference on Algebra and Coalgebra in Computer Science, J. Fiadeiro and J. Rutten, eds., Lecture Notes in Comput. Sci. 3629, Springer-Verlag, Berlin, 2005, pp. 128–142.
- [13] E.-E. DOBERKAT, *Semipullbacks for stochastic relations over analytic spaces*, Math. Structures Comput. Sci., 15 (2005), pp. 647–670.
- [14] C. J. HIMMELBERG, *Measurable relations*, Fund. Math., 87 (1975), pp. 53–72.
- [15] A. JOYAL, M. NIELSEN, AND G. WINSKEL, *Bisimulation from open maps*, Inform. and Comput., 127 (1996), pp. 164–185.
- [16] D. E. KOZEN, *Semantics of probabilistic programs*, J. Comput. Systems Sci., 22 (1981), pp. 328–350.
- [17] C. A. KUPKE, A. KURZ, AND Y. VENEMA, *Stone coalgebras*, Theoret. Comput. Sci., 327 (2004), pp. 109–134.
- [18] K. G. LARSEN AND A. SKOU, *Bisimulation through probabilistic testing*, Inform. and Comput., 94 (1991), pp. 1–28.
- [19] C. MORGAN, A. MCIVER, AND K. SEIDEL, *Probabilistic predicate transformers*, ACM Trans. Prog. Lang. Syst., 18 (1996), pp. 325–353.
- [20] L. S. MOSS AND I. VIGLIZZO, *Harsanyi Type Spaces and Final Coalgebras Constructed from Satisfied Theories*, Tech. report, Department of Mathematics, Indiana University, Bloomington, IN, 2004.
- [21] M. NARASIMHA, R. CLEAVELAND, AND P. IYER, *Probabilistic temporal logics via the modal mu-calculus*, in Proceedings of the FOSSACS'99, Amsterdam, The Netherlands, Lecture Notes in Comput. Sci. 1578, W. Thomas, ed., Springer-Verlag, Berlin, 1999, pp. 288–305.
- [22] P. PANANGADEN, *Stochastic Techniques in Concurrency*, Tech. report, BRICS, Department of Computer Science, University of Aarhus, Aarhus, Denmark, 1997.
- [23] P. PANANGADEN, *Probabilistic relations*, in Proc. PROBMIV '98, C. Baier, M. Huth, M. Kwiatkowska, and M. Ryan, eds., Electron. Notes Theor. Comput. Sci. 22, Elsevier, Amsterdam, 1999, pp. 59–74.
- [24] K. R. PARTHASARATHY, *Probability Measures on Metric Spaces*, Academic Press, New York, 1967.
- [25] J. J. M. M. RUTTEN, *Universal coalgebra: A theory of systems*, Theoret. Comput. Sci., 249 (2000), pp. 3–80. Special issue on modern algebra and its applications.
- [26] S. M. SRIVASTAVA, *A Course on Borel Sets*, Grad. Texts in Math. 180, Springer-Verlag, New York, 1998.
- [27] D. H. WAGNER, *A survey of measurable selection theorems*, SIAM J. Control Optim., 15 (1977), pp. 859–903.

SUBLINEAR GEOMETRIC ALGORITHMS*

BERNARD CHAZELLE[†], DING LIU[†], AND AVNER MAGEN[‡]

Abstract. We initiate an investigation of sublinear algorithms for geometric problems in two and three dimensions. We give optimal algorithms for intersection detection of convex polygons and polyhedra, point location in two-dimensional triangulations and Voronoi diagrams, and ray shooting in convex polyhedra, all of which run in expected time $O(\sqrt{n})$, where n is the size of the input. We also provide sublinear solutions for the approximate evaluation of the volume of a convex polytope and the length of the shortest path between two points on the boundary.

Key words. sublinear algorithms, approximate shortest paths, polyhedral intersection

AMS subject classifications. 68Q25, 68W05, 68W20, 68W40

DOI. 10.1137/S009753970444572X

1. Introduction. As an outgrowth of the recent work on property testing, the study of sublinear algorithms has emerged as a field unto itself, and great strides have been made in the context of graph and combinatorial problems [30]. Large geometric datasets often call for algorithms that examine only a small fraction of the input, but it is fair to say that sublinear computational geometry is still largely uncharted territory. If preprocessing is allowed, then, of course, this is an entirely different story [3, 23]. For example, checking whether a point lies in a convex 3-polyhedron can be done in logarithmic time with linear preprocessing. However, little of this technology is of any use with massive datasets, since examining the whole input—let alone preprocessing it—is out of the question. Sublinear algorithms have been given for dynamic problems [17] or in situations where a full multidimensional data structure is available [10]. There has also been work on geometric property testing, both in an approximate [11, 12, 18] and exact [24] setting.

In this paper, sublinearity is understood differently. The input is taken to be in any standard representation with no extra assumptions. For example, a planar subdivision or a polyhedron is given in classical edge-based fashion (e.g., doubly connected edge list (DCEL), winged-edge), with *no extra preprocessing*. This implies that we can pick an edge at random in constant time, but we cannot sample randomly among the neighbors of a given vertex in constant time. Our motivation is twofold: (i) we seek the minimal set of computational assumptions under which sublinearity is achievable; (ii) the assumptions should be realistic and nonrestrictive. Note, for example, that sublinear separation algorithms for convex objects are known [6, 15], but all of them require preprocessing, so they fall outside our model. Under these conditions one might ask whether there exist any interesting “offline” problems that can be solved in sublinear time. The answer is yes. Note that randomization is a necessity be-

*Received by the editors August 16, 2004; accepted for publication (in revised form) June 20, 2005; published electronically January 6, 2006. A preliminary version of this paper appeared in *Proceedings of the 35th Annual ACM Symposium on Theory of Computing*, San Diego, 2003, pp. 531–540. This work was supported in part by NSF grants CCR-998817 and ARO grant DAAH04-96-1-0181 and by the NEC Research Institute.

<http://www.siam.org/journals/sicomp/35-3/44572.html>

[†]Department of Computer Science, Princeton University, Princeton, NJ, 08544 (chazelle@cs.princeton.edu, dingliu@cs.princeton.edu).

[‡]Department of Computer Science, University of Toronto, Toronto, ON, M5S3G4 Canada (avner@cs.toronto.edu).

cause, in a deterministic setting, most problems in computational geometry require looking at the entire input. There has been some (but little) previous work on sub-linear geometric algorithms as we define them, specifically point location in two- and three-dimensional Delaunay triangulations of sets of random points [14, 27]. As far as we know, however, these are the only works that fall inside our model. Here is a summary of our results. In all cases, n denotes the input size, and all polyhedra are understood to be in \mathbf{R}^3 :

- an optimal $O(\sqrt{n})$ time algorithm for checking whether two convex polyhedra intersect, reporting an intersection point if they do and a separating plane if they do not;
- optimal $O(\sqrt{n})$ time algorithms for point location in planar convex subdivisions with $O(1)$ maximum face size and two-dimensional Voronoi diagrams, finding the nearest neighbor on a convex polyhedron, and ray shooting-type problems in convex polyhedra.

In contrast with property testing, it is important to note that our algorithms never err. All the algorithms are of Las Vegas type, and randomization affects the running time but not the correctness of the output.¹ Devroye, Mücke, and Zhu [14] showed that a simple technique for point location in two-dimensional Delaunay triangulations, namely random sampling then walking from the nearest sample to the query, has expected running time (roughly) $O(n^{1/3})$ for n random input points and a random query. This does not contradict the optimality of our $O(n^{1/2})$ bound because the points must be chosen randomly in [14].

We also consider optimization problems for which approximate solutions are sought. We give

- an $O(\varepsilon^{-1}\sqrt{n})$ time algorithm for approximating the volume of a convex polytope with arbitrary relative error $\varepsilon > 0$;
- an $O(\varepsilon^{-5/4}\sqrt{n}) + f(\varepsilon^{-5/4})$ time algorithm for approximating the length of the shortest path between two points on the boundary of a convex polyhedron with arbitrary relative error $\varepsilon > 0$. Here, $f(n)$ denotes the complexity of the exact version of problem. This implies that the complexity of our algorithm is $O(\sqrt{n})$ for any fixed $\varepsilon > 0$.

The shortest path problem for polyhedral surfaces has been extensively studied, drawing its motivation from applications in route planning, injection molding, and computer assisted surgery [1, 21, 26]. In the convex case (the one at hand), an $O(n^3 \log n)$ algorithm was given by Sharir and Schorr [32], later improved by Mitchell, Mount, and Papadimitriou [25] to $O(n^2 \log n)$ and by Chen and Han [7] to $O(n^2)$; therefore, it is known that $f(n) = O(n^2)$. More recently, Kapoor [22] has announced a proof that $f(n) = O(n \log^2 n)$, which would make our algorithm run in time $O(\varepsilon^{-5/4}\sqrt{n})$. This improves on Agarwal et al.'s algorithm [2], which runs in $O(n \log \varepsilon^{-1} + \varepsilon^{-3})$ time for any $\varepsilon > 0$.

Our method makes progress on an important geometric problem of independent interest.

- Given a convex polytope P of n vertices, how many vertices must an enclosing polytope Q have if it is to approximate any (large enough) shortest path on ∂P with relative error at most ε ? We reduce to $O(\varepsilon^{-5/4})$ the best previous bound of $O(\varepsilon^{-3/2})$, due to Agarwal et al. [2].

¹Throughout this paper, unless specified otherwise, the running times are understood in the expected sense.

A Flavor of the Techniques. As a warmup exercise, consider the classical *successor searching* problem: Given a sorted (doubly linked) list of n keys and a number x , find the smallest key $y \geq x$ (the *successor* of x) in the list or report that none exists. It is well known that this smallest key can be found in $O(\sqrt{n})$ expected time [19]. For this, we choose \sqrt{n} list elements at random and find the predecessor and successor of x among those. (Perhaps only one exists.) This provides an entry point into the list, from which a naive search takes us to the successor. To make random sampling possible, we may assume that the list elements are stored in consecutive locations (say, in a table). However—and this is the key point—no assumption is made on the ordering of the elements in the table. (Otherwise we could do a binary search.)

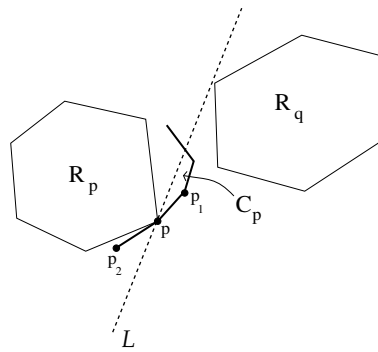
LEMMA 1.1. *Successor searching can be done in $O(\sqrt{n})$ expected time per query, which is optimal.*

Proof. For $i \geq 1$, let Q_i be the set of all elements that are at distance at most i away from the answer on the list (in either direction). Let $P_{>i}$ be the probability of not hitting Q_i after \sqrt{n} random choices of the list elements. The expected distance of the answer to its nearest neighbor in the random sample is $\sum_{i \geq 1} i(P_{>i-1} - P_{>i}) = \sum_{i \geq 0} P_{>i}$. This sum is upper bounded by $\sqrt{n} \sum_{c \geq 0} P_{>c\sqrt{n}} \leq \sqrt{n} \sum_{c \geq 0} (1 - c/\sqrt{n})^{\sqrt{n}} = \sqrt{n} \sum_{c \geq 0} 2^{-\Omega(c)} = O(\sqrt{n})$. This immediately implies that the expected time of the algorithm is $O(\sqrt{n})$.

For the lower bound, we use Yao’s minimax principle [33]. We fix a distribution on the input, and we lower-bound the expected complexity of any deterministic algorithm. We then have the same lower bound for randomized algorithms. The input is a linked list containing the numbers 1 through n in sorted order. In our model, the list is represented by a table $T[1 \dots n]$, with the i th element in the list stored in location $\sigma(i)$ of the table; hence, $T[\sigma(i)] = i$. The input distribution is formed by choosing the permutation σ uniformly from the symmetric group on n elements. In other words, all permutations are equally likely. The query is set to be n . In other words, the problem is to locate the last element in the list. A deterministic algorithm can be modeled as a sequence of steps of the following form: (A) pick a location $T[k]$ already visited and look up the next (or previous) item, i.e., $T[\sigma(i \pm 1)]$, where $k = \sigma(i)$; (B) compute a new index k and look up $T[k]$. Each step may involve the consideration of every piece of information gathered so far. In particular, in a B-step we may not consult either one of the adjacent items in the list before computing k (unless, of course, these items were visited earlier). In this way, $\sigma^{-1}(k)$ of a B-step is equally likely to lie anywhere in the portion of the list still unvisited. For this reason, after a A-steps and b B-steps, there is a probability at least $(1 - \frac{\sqrt{n+a+b}}{n})^b$ that none of the last \sqrt{n} elements in the list has been visited in a B-step. Right after the last B-step, either the total number of A- and B-steps exceeds \sqrt{n} or, with constant nonzero probability, at least \sqrt{n} A-steps (some of which may have already been taken) are required to reach the last element in the list. This immediately implies that the expected time of any deterministic algorithm is $\Omega(\sqrt{n})$. \square

We can generalize these ideas to polygon intersection. Given two convex polygons P and Q , with n vertices each, determine whether they intersect or not and, if they do, report one point in the intersection. We assume that P and Q are given by their doubly linked lists of vertices (or edges) such that each vertex points to its predecessor and successor in clockwise order. As in successor searching, we assume that the two lists are stored in two tables to allow random sampling.

Choose a random sample of r vertices from each polygon, and let $R_p \subseteq P$ and $R_q \subseteq Q$ denote the two corresponding convex hulls. By two-dimensional linear pro-

FIG. 1. *Intersecting two convex polygons.*

gramming, we can test R_p and R_q for intersection without computing them explicitly. This can be done probabilistically (or even deterministically) in linear time. There are many ways of doing that (see [5] for references). It is easy to modify the algorithm (of, say, [31]) so that in $O(r)$ time it reports a point in the intersection of R_p and R_q if there is one (in which case we are done) and a bitangent separating line \mathcal{L} otherwise (Figure 1). Let p be the vertex of R_p in \mathcal{L} , and let p_1, p_2 be its two adjacent vertices in P . We define a polygon C_p as follows. If neither p_1 nor p_2 is on the R_q side of \mathcal{L} , then C_p is the empty polygon. Otherwise, by convexity exactly one of them is (say, p_1). We walk along the boundary of P starting at p_1 , away from p , until we cross \mathcal{L} again. This portion of the boundary, clipped by the line \mathcal{L} , forms the convex polygon C_p . A similar construction for Q leads to C_q .

It is immediate that $P \cap Q \neq \emptyset$ if and only if P intersects C_q or Q intersects C_p . We check the first condition and, if it fails, check the second one. We restrict our explanation to the case of $P \cap C_q$. First, we check whether R_p and C_q intersect, again using a linear time algorithm for a linear program (LP), and return with an intersection point if they do. Otherwise, we find a line \mathcal{L}' that separates R_p and C_q and, using the same procedure as described above, we compute the part of P , denoted C'_p , on the C_q side of \mathcal{L}' . Finally, we test C'_p and C_q for intersection in time linear on their sizes, using an LP or any other straightforward linear-time algorithm for intersection detection of convex polygons. Correctness is immediate. The running time is $O(r + |C_p| + |C'_p| + |C_q| + |C'_q|)$. We can prove that $\mathbf{E}|C_p| = O(n/r)$. (The three-dimensional case discussed below will subsume this result, so there is no need for a proof now.) Similarly, $\mathbf{E}|C'_p| = \mathbf{E}|C_q| = \mathbf{E}|C'_q| = O(n/r)$. The overall complexity of the algorithm is $O(r + n/r)$, and choosing $r = \lfloor \sqrt{n} \rfloor$ gives the desired bound of $O(\sqrt{n})$.

To show optimality, consider the following distributions on pairs of polygons. One polygon is fixed, convex, and nondegenerate with one vertex in the origin and all other vertices below the x -axis. The other polygon (also convex and nondegenerate) has $n-1$ vertices above the x -axis, and one vertex, p , in the origin or in $(0, \delta)$, where δ is a positive number small enough so that p is the lowest vertex of the polygon. Moreover, the edges of this polygon are randomly ordered in the edge table. Clearly, these two polygons intersect if and only if p is in the origin. Since nothing in the structure of the input except the geometry of p reveals whether it is indeed the origin, any algorithm that detects intersection must have access to p . Now recall that the only operations allowed are the random sampling of edges and edge-traversing via links, which means

that, as in Lemma 1.1, an expected time of $\Omega(\sqrt{n})$ is needed to access p . Optimality of subsequent results follows these lines very closely and shall not be proved again. We have the following.

THEOREM 1.2. *To check whether two convex n -gons intersect can be done in $O(\sqrt{n})$ time, which is optimal.*

To put Theorem 1.2 in perspective, recall that the intersection of two convex polygons can be determined in logarithmic time if the vertices are stored in an array in cyclic order [6]. The key point of our result is that, in fact, a linked list is sufficient for sublinearity. Similarly, if polyhedra are preprocessed à la Dobkin and Kirkpatrick, then fast intersection detection is possible [15]. What we show below is that sublinearity is achievable even with no preprocessing at all. Again, we use a two-stage process: In the first stage we break up the problem into r subproblems of size roughly n/r and then identify which ones actually need to be solved; in the second stage we solve these subproblems in standard (i.e., nonsublinear) fashion. Their number is constant, and hence the square root complexity. What prevents us from solving these subproblems recursively is the model’s restriction to *global* random sampling. In other words, one can sample efficiently for the main problem but *not* for the subproblems.

2. Convex polyhedral intersections. Given two n -vertex convex polyhedra P and Q in \mathbf{R}^3 , the problem is to determine whether or not they intersect: If they do, then we should report a point in the intersection; otherwise, we should report a plane that separates them. We assume that a convex polyhedron is given in any classical edge-based fashion (e.g., DCEL, winged-edge) but with no extra preprocessing. The main structure is a table of edges that allows us to pick an edge at random in constant time. There are also two tables for vertices and faces. Moreover, these tables are interconnected via pointers to make various local operations possible. For example, each edge points to its two vertices and two adjacent faces. It also points to its predecessor and successor edges in its two adjacent faces. Such a structure is a standard representation for convex polyhedra in computational geometry. It allows us to traverse a portion of a convex polyhedron in a local fashion and in time linear in the number of edges visited.

Choose a random sample of $r = \lfloor \sqrt{n} \rfloor$ edges from P and Q , and let R_p and R_q denote the convex hulls of these random edges in P and Q , respectively. We do not compute R_p and R_q explicitly but merely use their vertices to get an LP as described in the last section for the case of polygons. We use this LP to detect the intersection of R_p and R_q in $O(r)$ time by invoking a linear-time algorithm for low-dimensional linear programming. We stop with a point of intersection if there is one. Otherwise, we find a separating plane \mathcal{L} that is tangent to both R_p and R_q . It is important to choose the plane \mathcal{L} in a canonical fashion. To do that, we set up the LP so as to maximize, say, the coefficient α in the equation² $\alpha x + \beta y + \gamma z = 1$ of \mathcal{L} .

Next, choose a plane π normal to \mathcal{L} and consider projecting P and Q onto it. (Of course, we do not actually do it.) Let p be a vertex of R_p in \mathcal{L} (there could be two of them, but not more, if we assume general position between P and Q), and let p^* be its projection onto π . We also project the neighbors of p in P onto π and get $p_1^*, p_2^*, \dots, p_k^*$. In other words, they are the set of vertices adjacent to p^* in the projection of P onto π . We test to see if any of them is on the R_q side of \mathcal{L} and identify one such point, p_1 , if the answer is yes (more on that below). If none of them

²With perturbation techniques, we can always assume general position, and hence avoid having a solution passing through the origin. We will also assume that the relative position of P and Q is general.

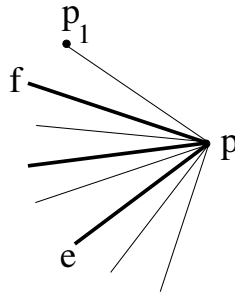


FIG. 2. The edges of P incident to p ; the thick lines form the random sample.

is on the R_q side, then we define C_p to be the empty polyhedron. This is because in this case, P is completely on the other side of \mathcal{L} . Otherwise, we construct the portion of P , denoted C_p , that lies on the R_q side of \mathcal{L} . Note that C_p is a convex polytope, not just the boundary of P cut off by \mathcal{L} . We compute C_p by using a standard flooding mechanism. Beginning at p_1 , we perform a depth-first search through the facial structure of P , restricted to the relevant side of \mathcal{L} . Because C_p is convex, the edges form a single connected component, so we never need to leave C_p . This allows us to build the entire facial representation of C_p in time proportional to its number of edges. From then on, the algorithm has the same structure as its polygonal counterpart; i.e., we compute C_p, C'_p, C_q, C'_q and perform the same sequence of tests.

The question is now, How do we find p_1 (if it exists)? To simplify the analysis, once we have p , we resample by picking r edges in P at random; let E be the subset of those incident to p . To find p_1 , we project on π all of the edges of E . If there exists an edge of E that is on the R_q side of \mathcal{L} , then we identify its endpoint as p_1 . Otherwise, all the edges of E lie on one side of \mathcal{L} . We then identify the two extreme ones (e and f in Figure 2); being extreme means that all the other projected edges of E lie in the wedge between e and f in π . Assume that e and f are well defined and distinct. Consider the cyclic list V of edges of P incident to p . The edges of E break up V into blocks of consecutive edges. It is not hard to prove that pp_1 lies in a block starting or ending with e or f if such a p_1 (as defined above) exists. So, we examine each of these relevant blocks (at most four) exhaustively. If e and f are not both distinct and well defined, we may simply search for p_1 by checking every edge of P incident to p .

THEOREM 2.1. *Two convex n -vertex polyhedra in \mathbf{R}^3 can be tested for intersection in $O(\sqrt{n})$ time; this is optimal.*

Proof. Optimality was already discussed in the polygonal case, and correctness follows from elementary convex geometry, so we limit our discussion to the complexity of the algorithm. Because of the resampling, the expected sizes of the blocks next to e and f (or, alternatively, the expected size of the neighborhood of p if the blocks are not distinct) are $O(n/r)$, so the running time is $O(r + n/r + \mathbf{E}(|C_p| + |C_q|))$, where $|C_p|$ (resp., $|C_q|$) denotes the number of edges of C_p (resp., C_q). We may exclude the other two terms $|C'_p|$ and $|C'_q|$, since our upper bound on $\mathbf{E}(|C_p| + |C_q|)$ will apply to them as well. Here is how to bound $\mathbf{E}(|C_p| + |C_q|)$ by $O(n/r)$.

We modify the sampling distribution a little. Then we argue that reverting back to the original setting does not change the asymptotic value of the upper bound. The modification is twofold: (i) we view $P \cup Q$ as a multiset M where each vertex appears as

many times as its number of incident edges; (ii) R_p and R_q are formed by picking each point of M independently with probability r/n and assigning points of P to R_p and points of Q to R_q . With respect to the modified distribution, $|C_p|+|C_q|$ is proportional to the number of constraints in M that violate the LP $\mathcal{P}(R_p, R_q)$ used to define \mathcal{L} (with each point of R_p and R_q defining a linear constraint). For technical reasons, we need to perturb M slightly to make it in general position. Specifically, we move each point of M away from its corresponding vertex in P or Q by an infinitesimal random amount, along the corresponding edge of P or Q . After this random perturbation, the size of the violation set of $\mathcal{P}(R_p, R_q)$ can only increase. To see this, note that constraints in M that violate $\mathcal{P}(R_p, R_q)$ before the perturbation continue to do so because the perturbation is infinitesimal; and constraints in M that lie on \mathcal{L} before the perturbation may violate the LP after the perturbation.

To bound the expected size of the violation set, we apply a result proved by Gärtner and Welzl [19] (and also by Clarkson [9]). Following the notation of the “Sampling Lemma” in [19], we let the ground set S be the (perturbed) set M . After sampling a set of points R from M randomly, we set up an LP $\alpha x + \beta y + \gamma z = 1$ that separates R_p and R_q . Note that in this LP, $\alpha, \beta,$ and γ are variables and (x, y, z) is a point in R . This LP is set up so as to maximize the variable α , whose optimum value is a function of R . We set the function ϕ in [19] to be α . Under this setting, the extreme elements of R_p and R_q are their vertices on the optimum separating plane (as implied by the above LP). Since M is in general position, any R_p and R_q have three extreme elements. The Sampling Lemma in [19] then implies that the expected size of the violation set is at most $3(n - r)/(r + 1) = O(n/r)$.

Let \mathcal{D} be the original distribution (the one used by the actual algorithm) with r replaced by $13r$. Of course, this scaling has no asymptotic effect on the upper bound for $\mathbf{E}(|C_p| + |C_q|)$. We define an intermediate distribution \mathcal{D}_1 by going through each edge (u, v) of $P \cup Q$ twice, selecting it with probability r/n , and then throwing into the sample both u and v , provided that the edge (u, v) has not yet been selected. (Note that this implies that u and v are kept out with probability $(1 - r/n)^2$.) There are at most $6n$ edges in P and Q , so the probability that a sample from \mathcal{D}_1 is of size less than $13r$ is overwhelmingly high. Since all equal-size subsets of edges are equally likely to be chosen, $\mathbf{E}_{\mathcal{D}}(|C_p| + |C_q|)$ is nonincreasing with the sample size, and so $\mathbf{E}_{\mathcal{D}}(|C_p| + |C_q|) = O(\mathbf{E}_{\mathcal{D}_1}(|C_p| + |C_q|))$. Let \mathcal{D}_2 denote the modified distribution used in the calculations. Observe that \mathcal{D}_2 is derived from \mathcal{D}_1 by picking only u if (u, v) is chosen the first time it is considered for selection and then only v if it is picked the second time around. By monotonicity, we then have $\mathbf{E}_{\mathcal{D}_1}(|C_p| + |C_q|) = O(\mathbf{E}_{\mathcal{D}_2}(|C_p| + |C_q|))$. This proves that the $O(n/r)$ bound holds in the original distribution used by the algorithm.

Recall that the running time is $O(r + n/r + \mathbf{E}(|C_p| + |C_q|))$, which is $O(r + n/r)$ by the above analysis. For $r = \lfloor \sqrt{n} \rfloor$ it is $O(\sqrt{n})$. \square

When the two convex polyhedra intersect, the algorithm reports a point in the intersection. On the other hand, when they are disjoint, we can report a plane that separates them. Here is a brief description on how to do that. Note that we cannot simply return a separating plane for C_p and C'_q (or C'_p and C_q) because it is not necessarily separating for P and Q . Instead, we resort to geometric duality to compute the desired plane in expected $O(\sqrt{n})$ time. In a standard geometric duality transform, a vertex in the primal space is mapped to a plane in the dual space and vice versa. Moreover, the upper (resp., lower) hull of a convex polyhedron is transformed to a lower (resp., upper) envelope [13]. When P and Q are disjoint, at least one of the following must be true: (1) there exists a plane above the upper hull of P and below

the lower hull of Q ; (2) there exists a plane below the lower hull of P and above the upper hull of Q . Since they are symmetric, it suffices to consider the first one. By duality, such a plane dualizes to a point in the common intersection of an upper and a lower envelope, which is itself a convex polyhedron. Although this polyhedron is not available explicitly, we have access to its geometric features (vertices, edges, etc.) in constant time via the corresponding features in the primal space. Hence we can apply the above algorithm to find, in $O(\sqrt{n})$ time, an intersection point which is the dual of a separating plane for P and Q .

It is important to note that an alternative to the above approach can be found in a general scheme to solve the constant-dimensional LP in linear time due to Clarkson [9]. Clarkson suggested a randomized algorithm that finds a set of constraints of expected size $O(\sqrt{n})$ (or, in general, $O(d\sqrt{n})$, where d is the dimension) that contains a “basis,” that is, a minimal set of constraints that determines the problem. Our approach is somewhat similar to that schema. Of course, there are details to be filled as to how exactly this set may be computed in time $O(\sqrt{n})$. (Clarkson’s algorithm as it is would be a linear time algorithm.) In particular, an $O(\sqrt{n})$ -time method of finding the violating subpolyhedron (like the one we proposed) must still be used in order to implement the alternative approach of Clarkson efficiently enough.

3. Ray shooting applications. Given a convex polyhedron P with n vertices and a directed line ℓ in \mathbf{R}^3 , the ray shooting problem asks for the point on (the boundary of) P hit by ℓ if it exists. We apply essentially the same techniques as in convex polyhedral intersection to ray shooting and solve it in expected $O(\sqrt{n})$ time. Choose a random sample of $\lfloor \sqrt{n} \rfloor$ edges from P , and let R_p denote the convex hull of these edges. We first use an LP to detect intersection of R_p and ℓ in time $O(\sqrt{n})$. There are two cases. If R_p and ℓ do not intersect, we get a plane \mathcal{L} that separates them and passes through a vertex q of R_p . Starting from q we construct the intersection C_p of P with the halfspace bounded by \mathcal{L} that contains ℓ . We already explained how to do that in the previous section. Finally, we solve ray shooting for C_p and ℓ . Now suppose that R_p and ℓ intersect. We first find the point p on R_p hit by ℓ in time $O(\sqrt{n})$. We cannot afford to compute an explicit representation of R_p in time $\Omega(\sqrt{n} \log n)$. To find p we again use an LP. We can assume that ℓ is the positive x -axis by rotating the coordinate system. Of course, we do not rotate the whole polytope P . Instead, we maintain such a rotation transform implicitly. In other words, whenever we need a geometric feature (vertex, edge, etc.) of P after the rotation, we compute it from its corresponding feature on the original input in constant time. Finding p is equivalent to finding a plane \mathcal{L} such that (1) all vertices of R_p are on one side of \mathcal{L} (the side that contains $(+\infty, 0, 0)$); (2) the intersection point of \mathcal{L} with the x -axis has its x -coordinate as large as possible. In fact, p is that intersection point. It is straightforward to formulate this problem as a three-dimensional LP and solve it in time $O(\sqrt{n})$. In particular, to ensure (2) above we minimize the coefficient α in the equation $\alpha x + \beta y + \gamma z = 1$ for \mathcal{L} . Once we have \mathcal{L} and p , we construct C_p as before and solve the problem for C_p and ℓ . Essentially the same analysis as the proof of Theorem 2.1 shows that the expected size of C_p is $O(\sqrt{n})$. We thus have the following.

THEOREM 3.1. *Given a convex polyhedron with n vertices and a directed line, we can compute their intersection explicitly in optimal $O(\sqrt{n})$ time.*

This sublinear time algorithm for ray shooting towards a convex polyhedron gives us useful ammunition for all sorts of location problems.

Given the Delaunay triangulation \mathcal{T} of a set S of n points in the plane and a

query point q , consider the problem of locating q , i.e., retrieving the triangle of \mathcal{T} that contains it. The Delaunay triangulation can be given in any classical edge-based data structure (e.g., DCEL), as long as it supports $O(1)$ time access to a triangle from a neighboring triangle. We use the close relationship between Delaunay triangulations and convex hulls given by the mapping $h : (x, y) \mapsto (x, y, x^2 + y^2)$. As is well known, the Delaunay triangulation of S is facially isomorphic to the lower hull of $h(S)$ (i.e., the part of the convex hull that sees $z = -\infty$). In this way, point location in \mathcal{T} is equivalent to ray shooting towards the convex hull, where the ray originates from the query point q and shoots in the positive z -direction. Obviously, any facial feature of the convex hull can be retrieved in constant time from its corresponding feature in the Delaunay triangulation. (The one exception is the set of faces outside the lower hull: we can simplify matters by adding a dummy vertex to the hull at $z = \infty$.)

The same argument can be used for point location in Voronoi diagrams. Recall that each point (p_x, p_y) is now lifted to the plane $Z = 2p_xX + 2p_yY - (p_x^2 + p_y^2)$, which is tangent to the paraboloid $Z = X^2 + Y^2$. The Voronoi diagram of S is isomorphic to the lower envelope of the arrangement formed by the n tangent planes. Note that any vertex (resp., edge) of the envelope can be derived in constant time from the three (resp., two) faces incident to the corresponding vertex (resp., edge).

THEOREM 3.2. *Point location in the Delaunay triangulation or Voronoi diagram of n points in the plane can be done in optimal $O(\sqrt{n})$ time.*

Observe that algorithms for computing a Delaunay triangulation or a Voronoi diagram often supply an efficient point location data structure as a by-product, and thus sublinear time point location in Delaunay triangulations or Voronoi diagrams may be of lesser interest. However, our algorithm is still useful when the triangulation/diagram is huge and we cannot afford to store it together with the point location structure. Our algorithm for point location in Delaunay triangulations also has its limitations: It works only because of the known correspondence between a Delaunay triangulation and a special convex polyhedron. It cannot perform point location in arbitrary planar triangulations. In the next section, we use a different method to achieve sublinear time point location in arbitrary triangulations or convex subdivisions with $O(1)$ maximum face size.

We consider the following problem, which will arise in our subsequent discussion of volume approximation and shortest path algorithms. Given a convex polyhedron P with n vertices and a point q , let $n_P(q)$ denote the (unique) point of P that is closest to q . Of course, we can assume that q does not lie inside P , which we can test by using the previous algorithm. To compute $n_P(q)$ we extract a sample polyhedron R_p of size \sqrt{n} (as we did before) and find $n_{R_p}(q)$. Since we just have a collection of vertices of R_p instead of its full facial representation, it is not obvious how to find $n_{R_p}(q)$ in time $O(\sqrt{n})$. For this purpose, we express this problem as an *LP-type* problem and solve it using the method in [5] (see Chapter 8). A reformulation of the problem would be to seek the plane \mathcal{L} that separates q from the vertices of R_p and maximizes the distance from q to it. To apply the method in [5], we view each vertex of R_p as a constraint. We also check that all the assumptions (i.e., monotonicity, locality, violation test, and range space oracle) needed to solve this problem efficiently hold. See [5] for details. Thus we get \mathcal{L} in time $O(\sqrt{n})$: it is tangent to R_p at $n_{R_p}(q)$ and normal to the segment $qn_{R_p}(q)$. Next, we compute the intersection C_p of P with the halfspace bounded by \mathcal{L} that contains q . Again, a similar analysis shows that the expected size of C_p is $O(\sqrt{n})$. Obviously, $n_P(q) = n_{C_p}(q)$, so we can finish the work by exhaustive search in C_p .

THEOREM 3.3. *Given a convex polyhedron P with n vertices and a point q , the nearest neighbor of q in P can be found in $O(\sqrt{n})$ time.*

We can compute a related function by similar means. Given a directed line ℓ , consider an orthogonal system of coordinates with ℓ as one of its axes (in the positive direction), and define $\xi_P(\ell)$ to be any point of P with maximum ℓ -coordinate. If we choose a point q at infinity on ℓ , then $\xi_P(\ell)$ can be chosen as $n_P(q)$, and so we can apply Theorem 3.3.

Another function we can compute in this fashion maps a plane \mathcal{L} and a direction ℓ in \mathcal{L} to the furthest point of P in \mathcal{L} along ℓ : in other words, $\xi_P(\mathcal{L}, \ell) = \xi_{P \cap \mathcal{L}}(\ell)$. Again, the nonobvious part is computing $\xi_{R_p}(\mathcal{L}, \ell)$ in time $O(\sqrt{n})$ for a sample polytope R_p . As in the case of ray shooting, we can assume without loss of generality that \mathcal{L} is the xy -plane and ℓ is the positive x -direction. Finding $\xi_{R_p}(\mathcal{L}, \ell)$ is the same as finding a plane \mathcal{L}' such that (1) all vertices of R_p are on one side of \mathcal{L}' (the side that contains $(-\infty, 0, 0)$); (2) \mathcal{L}' is parallel to the y -axis; (3) the intersection point of \mathcal{L}' with the x -axis has its x -coordinate as small as possible. We solve this problem in time $O(\sqrt{n})$ by formulating it as a three-dimensional LP. Other parts of the algorithm (e.g., constructing C_p) and its analysis are similar to other problems discussed in this section. We summarize our results.

THEOREM 3.4. *Given a convex polyhedron P with n vertices, a directed line ℓ , and a plane π , the points $\xi_P(\ell)$ and $\xi_P(\pi, \ell)$ can be found in $O(\sqrt{n})$ time.*

4. Point location in convex subdivisions. Given a convex planar subdivision \mathcal{S} with n edges and a query point q , the point location problem asks for the face of \mathcal{S} that contains q . In the previous section, we provided an $O(\sqrt{n})$ -time point location algorithm where \mathcal{S} is a Delaunay triangulation or a Voronoi diagram. Devroye, Mücke, and Zhu [14] also showed that a simple “walk-through” technique locates a query point in the Delaunay triangulation of n random points in the plane in expected (roughly) $O(n^{1/3})$ time. Here we show that a slight variation of the walk-through technique actually locates a query point in *any* planar triangulation (not necessarily Delaunay or formed by random points) in expected $O(\sqrt{n})$ time, which is optimal. Our algorithm generalizes to planar subdivisions with $O(1)$ maximum face size. We also give a simple argument showing an $\Omega(n)$ lower bound for point location in subdivisions with large faces.

THEOREM 4.1. *Point location in an n -edge convex planar subdivision with $O(1)$ maximum face size can be done in optimal $O(\sqrt{n})$ time.*

Proof. First, we consider the case of a triangulation. For an edge e in the triangulation and a query point q , we use q_e to denote the nearest neighbor of q on e . It is natural to define the Euclidean distance between q and e as $|qq_e|$. We start by sampling \sqrt{n} edges of the triangulation at random. Let e be the edge in the random sample that has the smallest Euclidean distance to q . We walk from q_e toward³ q by traversing all triangles crossed by qq_e one by one. Given any edge-based representation of the triangulation (such as DCEL), it takes constant time to traverse from one triangle to the next. We stop at the triangle that contains q and output it as the answer.

The running time (besides the sampling stage) is proportional to the number of triangles crossed by qq_e . Thus it suffices to show that the expected number of

³It is important to walk towards q from its nearest neighbor on the nearest edge. In contrast, previous algorithms [27] either walk from an endpoint (or the midpoint) of the nearest edge or sample by vertices and walk from the nearest sample vertex. These algorithms do not have sublinear expected running time for arbitrary triangulations.

triangulation edges crossed by qq_e is $O(\sqrt{n})$. For this, we rank each edge according to its Euclidean distance to q . Since the rank of every edge crossed by qq_e is smaller than that of e , the number of edges crossed by qq_e is at most the rank of e . The claimed time bound then follows from the fact that the smallest rank of \sqrt{n} random edges has expectation $O(\sqrt{n})$. It is straightforward to generalize this algorithm to planar subdivisions with $O(1)$ maximum face size. \square

What if the subdivision has large faces: Is sublinear time point location still possible? The answer is no.

THEOREM 4.2. *There exists an n -edge planar subdivision such that any randomized algorithm for point location in this subdivision has expected running time $\Omega(n)$.*

Proof. Consider the following problem first: We are given a doubly linked list of numbers. We know exactly one of them is nonzero and want to find out that special number. We can use an argument similar to the proof of Lemma 1.1 to show that any randomized algorithm has to spend $\Omega(n)$ expected time on this problem. Returning to point location, consider a rectangle with corners $(-1, 0)$, $(-1, n + 1)$, $(1, 0)$, and $(1, n + 1)$. By breaking its two vertical sides into $n + 1$ unit length segments, we get a face with $2n + 4$ edges. Finally, we pick an integer i from 1 to n and add a horizontal edge from $(-1, i)$ to $(1, i)$. This gives us a two-face subdivision. Given the query point $(0, (n + 1)/2)$, a deterministic algorithm must find the horizontal edge in the middle to locate the query correctly, and the only way to do that is through a visit to one of its four adjacent edges. This is similar in spirit to the list-checking problem considered above. In other words, in both problems we try to find one of $O(1)$ special elements in a list⁴ of size $\Theta(n)$. We thus get the same lower bound of $\Omega(n)$. \square

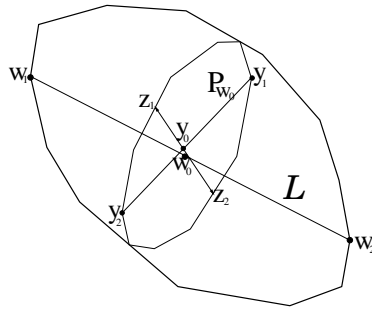
5. Volume approximation. We seek to approximate the volume of a convex polytope P . We proceed in two stages. First, we compute a large enough enclosed ellipsoid, which we use to rescale P affinely. This is intended to make P round enough so that good Hausdorff distance approximation yields good volume approximation. Second, we use a standard construction of Dudley [16] to find, via the methods of the previous section, an enclosing polytope of $O(1/\varepsilon)$ vertices whose boundary is at Hausdorff distance at most ε from P .

STAGE 1. We begin by computing, in $O(\sqrt{n})$ time, a polytope $P' \subseteq P$, such that $\text{vol}(P') \geq c_0 \text{vol}(P)$ for some constant $c_0 > 0$. Compute the six points $\xi_P(\ell)$ for $\ell = \pm x, \pm y, \pm z$. These points come in pairs, so let w_1, w_2 be the pair forming the largest distance. Given a point w on the line \mathcal{L} passing through w_1 and w_2 , let P_w denote the intersection of P with the plane through w that is orthogonal to \mathcal{L} . Let w_0 be the midpoint of $w_1 w_2$ (Figure 3). We first show that if S is a set of points in P_{w_0} such that

$$(1) \quad \text{area}(\text{conv}(S)) \geq c_1 \text{area}(P_{w_0})$$

for some constant $c_1 > 0$, then $\text{vol}(\text{conv}(S \cup \{w_1, w_2\})) \geq c_2 \text{vol}(P)$ for some other constant $c_2 > 0$. Therefore, we can take $P' = \text{conv}(S \cup \{w_1, w_2\})$ to achieve our goal. Indeed, assume we have such a set S . As a straightforward consequence of Pythagorean theorem, we find that $\text{diam}(P) \leq \sqrt{3} d(w_1, w_2)$; therefore, the orthogonal projection of P on \mathcal{L} is a segment $v_1 v_2 \supseteq w_1 w_2$ of length at most $\sqrt{3} d(w_1, w_2)$. This implies that, for any w in \mathcal{L} , $\text{area}(P_w) \leq 12 \text{area}(P_{w_0})$. To see why, observe that if, say, $w \in v_1 w_0$, then, by convexity, P_w is enclosed in the cone with apex w_2 and base

⁴In the point location problem there are two lists of vertical edges instead of one. This requires only a small modification of the argument.

FIG. 3. Approximating P from within.

P_{w_0} . Therefore, P_w lies in a copy of P_{w_0} scaled by at most $d(w, w_2)/d(w_0, w_2) \leq 2\sqrt{3}$, which proves our claimed upper bound on $\mathbf{area}(P_w)$. Of course, the same argument can be repeated if $w \in w_0v_2$. Since $\mathbf{vol}(P) = \int_{v_1}^{v_2} \mathbf{area}(P_w) dw$, we can conclude that the four quantities

$$\mathbf{vol}(P), \quad \mathbf{vol}(\mathbf{conv}(P_{w_0} \cup \{v_1, v_2\})),$$

$$\mathbf{vol}(\mathbf{conv}(P_{w_0} \cup \{w_1, w_2\})), \quad \mathbf{vol}(\mathbf{conv}(S \cup \{w_1, w_2\}))$$

are all equal up to within constant factors.

We now show how to find a set S satisfying (1). We essentially repeat in two dimensions what we did so far in three dimensions. Specifically, we take a, b to be two mutually orthogonal vectors both normal to \mathcal{L} , and let π be the plane spanned by a and b . We compute the four points (two pairs) $\xi_P(\pi, \ell)$ for $\ell = a, -a, b, -b$. Let y_1, y_2 be the more distant pair (analogous to w_1, w_2 before). Let y_0 be the midpoint of y_1, y_2 , and let segment ℓ_{y_0} be the intersection of P with the line in π orthogonal to y_1y_2 . We can find the two endpoints z_1, z_2 of ℓ_{y_0} using ray shooting. Using almost the same argument as the one showing that $\mathbf{conv}(P_{w_0} \cup \{w_1, w_2\})$ has a volume proportional to $\mathbf{vol}(P)$, we get that the quadrilateral with vertex set $S = \{y_1, y_2, z_1, z_2\}$ has an area proportional to $\mathbf{area}(P_{w_0})$, and thus satisfies (1). We comment that a similar approach to the one we described above was used by Barequet and Har-Peled [4]. The difference is that they approximate the volume of a convex polytope from outside by a bounding box, whereas we approximate it from within.

Let \mathcal{E} be the largest ellipsoid enclosed in $P' = \mathbf{conv}(\{y_1, y_2, z_1, z_2, w_1, w_2\})$, also known as the Löwner–John ellipsoid. It is computable in constant time within any fixed relative error by solving a constant-size quadratic program [20]. As is well known, its volume is at least $(1/\dim)^2$ times that of the enclosing polytope; therefore,

$$\mathbf{vol}(\mathcal{E}) \geq \frac{1}{9} \mathbf{vol}(P') \geq c \cdot \mathbf{vol}(P)$$

for some constant $c > 0$. Make the center of the ellipsoid the origin of the system of coordinates and use the ellipsoid's positive semidefinite matrix to rescale P . To do that, we consider the linear transformation that takes the ellipsoid into a ball of the same volume. Specifically, if $x^T A^T A x \leq 1$ is the equation of the ellipsoid, then we consider the transformation $T = A/(\det A)$. The polytope TP has the same volume as P , but it is *round*; namely, it contains a ball B of volume $\Omega(\mathbf{vol}(TP))$. Thus, we might as well assume that P has this property to begin with. Note that P is also

enclosed in a concentric ball B' that differs from B by only a constant-factor scaling. (If not, then TP would contain a point p so far away from B that the convex hull of p and B , although contained in P , would have volume much larger than $\text{vol}(B)$, and hence $\text{vol}(P)$, which would give a contradiction.) Finally, by rescaling we can also assume that P is enclosed in the unit ball and its volume is bounded below by a positive constant. By Theorems 3.1 and 3.4, all of the work in Stage 1 can be done in $O(\sqrt{n})$ time.

STAGE 2. We implement Dudley’s construction [16] of a convex polytope Q such that (i) $Q \supseteq P$; (ii) $Q \subset P_\varepsilon$, where P_ε is the Minkowski sum of P with a ball of radius ε ; (iii) Q has $O(1/\varepsilon)$ vertices. Dudley’s result was used constructively in [2]. The difference here is that our implementation is sublinear. We compute an $\sqrt{\varepsilon}$ -net on the unit sphere,⁵ and project this net down to ∂P , using the nearest neighbor function n_P as a projection map. Finally, we form Q as the intersection of the $O(1/\varepsilon)$ halfspaces bounded by the appropriate tangent planes passing through the vertices of the projected net. With suitable use of the nearest neighbor algorithm of Theorem 3.3, we can implement the entire construction in time $O(\varepsilon^{-1}\sqrt{n})$ for the projection construction (since the facial representations of P and TP are the same, the algorithm can use TP as though it had its full facial representation at its disposal) and $O(\varepsilon^{-1} \log \varepsilon^{-1})$ for intersecting the halfspaces needed to form Q . Since we can obviously assume that Q does not have more vertices than P , there is no need for ε to be smaller than, say, $1/n^2$. This implies that the entire construction time is dominated by $O(\varepsilon^{-1}\sqrt{n})$. (In fact, we can further reduce this running time to roughly $O(\varepsilon^{-1/2}\sqrt{n})$ by exploiting the fact that the $O(\varepsilon^{-1})$ nearest neighbor queries can be answered in a more efficient batch mode. Similarly, we can also get a slight improvement on the running time in Theorem 6.2.)

We now show that $\text{vol}(Q) = (1 + O(\varepsilon))\text{vol}(P)$. Recall that P is “sandwiched” between two concentric balls B and B' such that $\text{rad}(B') = 1$ and $\text{rad}(B) = \Omega(1)$. We may assume that B and B' are centered at the origin. Since $Q \subset P_\varepsilon$, we have $\text{vol}(Q - P) \leq \text{vol}(P_\varepsilon - P) \leq \text{area}(P_\varepsilon) \cdot 2\varepsilon \leq \text{area}(B_{1+2\varepsilon}) \cdot 2\varepsilon = O(\varepsilon)$, where $B_{1+2\varepsilon}$ is a ball centered at the origin with radius $1 + 2\varepsilon$. The upper bound on $\text{vol}(P_\varepsilon - P)$ is obtained by integration over thin shells of increasing area from ∂P to ∂P_ε . Since $\text{vol}(P) = \Omega(1)$, we then have $\text{vol}(Q) = (1 + O(\varepsilon))\text{vol}(P)$.

THEOREM 5.1. *Given any $\varepsilon > 0$, it is possible to approximate the volume of an n -vertex convex polytope with arbitrary relative error $\varepsilon > 0$ in time $O(\varepsilon^{-1}\sqrt{n})$.*

6. Approximate shortest paths. Given a convex polyhedron P with n vertices and two points s and t on its boundary ∂P , the problem is to find the shortest path between s and t outside the interior of P . It is well known that the shortest path lies on the boundary ∂P . In fact, it is easy to construct instances where any reasonable approximation of the shortest path on ∂P involves $\Omega(n)$ edges. This rules out sublinear algorithms, unless we are willing to follow paths outside of P . We show how to compute a path between s and t whose length exceeds the minimum by a factor of at most $1 + \varepsilon$ for any $\varepsilon > 0$.

Our algorithm relies on a new result of independent interest. Let $d_P(s, t)$ denote the length of the shortest path between s and t in ∂P . Given a point $v \in \partial P$, let H_v be the supporting plane of P at v (or any such plane if v is a vertex), and let H_v^+ denote the halfspace bounded by H_v that contains P . Given $\varepsilon > 0$, we say that a

⁵This is a collection of $O(\varepsilon^{-1})$ points on the sphere such that any spherical cap of radius $\sqrt{\varepsilon}$ contains at least one of the points.

convex polytope Q is an ε -wrapper of P if (c_0 is an absolute constant discussed below)

- (i) Q encloses P ;
- (ii) the Hausdorff distance between ∂P and ∂Q is at most $\varepsilon \operatorname{diam}(P)$;
- (iii) given any $s, t \in \partial P$ such that $d_P(s, t) \geq c_0 \operatorname{diam}(P)$, $d_{\widehat{Q}}(s, t) \leq (1 + \varepsilon)d_P(s, t)$,
where $\widehat{Q} = Q \cap H_s^+ \cap H_t^+$.

LEMMA 6.1. *Any convex 3-polytope has an ε -wrapper of size $O(1/\varepsilon)^{5/4}$ for any $\varepsilon > 0$.*

This result improves on the $O(1/\varepsilon)^{3/2}$ bound of Agarwal et al. [2]. The use of a wrapper is self-evident. First, we clip the polytope to ensure that $d_P(s, t) \geq c_0 \operatorname{diam}(P)$ (section 6.1). Next, we compute an ε -wrapper (section 6.2) and approximate the shortest path between s and t by computing the shortest path between the two points in $\partial \widehat{Q}$. This can be done in quadratic time by using an algorithm by Chen and Han [7]. The resulting path, which is of length $(1 + O(\varepsilon))d_P(s, t)$, can be shortened to $(1 + \varepsilon)d_P(s, t)$ by rescaling ε suitably. Note that in (iii) the condition on s and t being sufficiently far apart is essential. It is a simple exercise to show that no variant of a wrapper can accommodate all pairs (s, t) simultaneously. If $f(n)$ denotes the complexity of the *exact* version of problem, then we have the following.

THEOREM 6.2. *Given any $\varepsilon > 0$ and two points s, t on the boundary of a convex polytope P of n vertices, it is possible to find a path between s and t outside P of length at most $(1 + \varepsilon)d_P(s, t)$ in time $O(\varepsilon^{-5/4}\sqrt{n}) + f(\varepsilon^{-5/4})$.*

We refer the reader back to the introduction for a discussion of the implication of this result in view of the state-of-the-art on the function $f(n)$.

6.1. Computing short paths. Given two points $s, t \in \partial P$, our first task is to ensure that $d_P(s, t) \geq c_0 \operatorname{diam}(P)$ for some constant $c_0 > 0$. To do this, we first compute a value δ such that $\delta \leq d_P(s, t) \leq 8\delta$. We will substitute for P the intersection P' of P with a clipping box centered at s of side length 16δ . Obviously, the shortest paths between s and t relative to P and P' are identical. The only computational primitive we need is the nearest neighbor function of Theorem 3.3. Note that we need only this function relative to P (not to P'). In fact, we first use this function to compute a few sample points on ∂P (see section 6.2). We then discard sample points that are outside of the clipping box. The remaining points together with the clipping box are used to compute an ε -wrapper of P' .

To compute a constant-factor approximation for $d_P(s, t)$, we adapt an algorithm of Har-Peled [21] to our sublinear setting. All that is needed is an implementation of the following primitive: Given two rays r_1, r_2 from a fixed point $p \in P$, let H be the plane spanned by these two rays, and let C denote the two-dimensional cone in H wedged between r_1 and r_2 . Given an additional query ray $r \in H$ (not necessarily emanating from p), we need to compute $\xi_{C \cap P}(H, r)$. By Theorem 3.4, this can be done in $O(\sqrt{n})$ time.

6.2. The ε -wrapper construction. Assuming without loss of generality that $\operatorname{diam}(P) = 1$, it suffices to prove the following.

THEOREM 6.3. *Given any $\varepsilon > 0$ and a convex polytope P of n vertices with diameter 1, there exists a convex polytope Q with $O(\varepsilon^{-5/4})$ vertices such that (i) $Q \supseteq P$; (ii) the Hausdorff distance between ∂P and ∂Q is $O(\varepsilon)$; and (iii) given any $s, t \in \partial P$ such that $d_P(s, t) \geq c_0$ for some constant c_0 , $d_{\widehat{Q}}(s, t) \leq (1 + O(\varepsilon))d_P(s, t)$, where $\widehat{Q} = Q \cap H_s^+ \cap H_t^+$.*

We first show how to construct Q . Let S be a sphere of radius 2 centered at some arbitrary point in P . Draw a grid G of longitudes and latitudes on S , so that

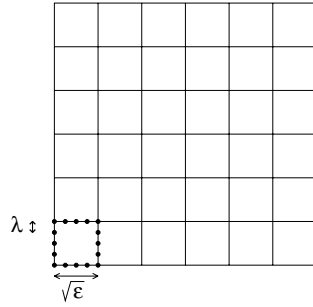


FIG. 4. The grid G .

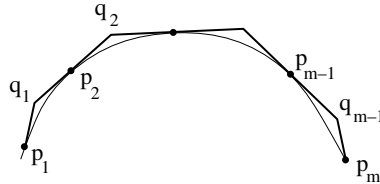


FIG. 5. The path σ .

each cell is of length $\sqrt{\epsilon}$ by $\sqrt{\epsilon}$ (with an exception made for the last latitude and longitude if $\sqrt{\epsilon}$ does not divide π). All lengths in this discussion are Euclidean, *except* in this case where the length of a circular arc refers to its corresponding angle. We choose a parameter $\lambda = \epsilon^{3/4}$ and subdivide each side of a cell into subarcs of length λ (Figure 4). In this way each cell has $O(\sqrt{\epsilon}/\lambda)$ vertices, and the whole construction defines a set V of $O(1/\lambda\sqrt{\epsilon})$ vertices. For each point $v \in V$, we compute $n_P(v)$, its nearest neighbor in ∂P , and define

$$(2) \quad Q = \bigcap \{ H_{n_P(v)}^+ \mid v \in V \}.$$

It is immediate from our choice of λ that Q has $O(\epsilon^{-5/4})$ vertices.⁶ Every point of the sphere S has at least one vertex of G at distance $O(\sqrt{\epsilon})$. By a result of Dudley [16], this implies part (ii) of Theorem 6.3. Since (i) is obvious, it remains for us to prove (iii).

Borrowing terminology from Agarwal et al. [2], we say that a pair (σ, \mathcal{H}) forms a *supported path* of P if $\sigma = p_1, q_1, p_2, q_2, \dots, q_{m-1}, p_m$ is a polygonal line disjoint from the interior of P and $\mathcal{H} = H_{p_1}, \dots, H_{p_m}$ is a sequence of supporting planes of P , such that $q_{i-1}p_i$ and p_iq_i both lie in H_{p_i} , with $q_0 = p_1$ and $q_m = p_m$ (Figure 5). For $0 < i < m$, the *folding angle* α_i at q_i is the dihedral angle of the wedge between H_{p_i} and $H_{p_{i+1}}$ (the one that does not contain P). The folding angle of σ is defined as $\alpha(\sigma) = \sum_{0 < i < m} \alpha_i$.

LEMMA 6.4 (Agarwal et al. [2]). *Given $s, t \in \partial P$, there exists a supported path σ of P with $O(1/\epsilon)$ edges, joining s and t , such that*

$$d_P(s, t) \leq |\sigma| \leq (1 + \epsilon)d_P(s, t) \quad \text{and} \quad \alpha(\sigma) = O(\epsilon^{-1/2}).$$

⁶To be precise, we actually want to compute an ϵ -wrapper for P' instead of P . For this, we need to remove from the point set V all vertices whose nearest neighbors in ∂P fall outside of the clipping box in section 6.1. We also need to clip Q by that box. We omit these minor details.

To help build intuition for the remainder of our discussion, it is useful to sketch the proof of the lemma. Mapping the grid G to P via the nearest neighbor function n_P creates a grid $n_P(G)$ on ∂P (with curved, possibly degenerate edges). It is convenient to think of P as a smooth manifold by infinitesimally rounding the vertices and edges. It does not much matter how we do that, as long as the end result endows each point $p \in \partial P$ with an (outward) unit normal vector η_p that is a continuous function of p . Note that in this way, for any $u \in S$, the vectors $un_P(u)$ and $\eta_{n_P(u)}$ are collinear, and the function n_P is a bijection. The fundamental property of the nearest neighbor function is that it is nonexpansive. We need only a weak version of that fact, which follows directly from Lemmas 4.3 and 4.4 in [16].

LEMMA 6.5 (Dudley [16]). *Given two points $p, q \in \partial P$, $|pq|$ and $\angle(\eta_p, \eta_q)$ are both in $O(|n_P^{-1}(p)n_P^{-1}(q)|)$.*

This implies that, for any two points $p, q \in \partial P$ in the same cell of the mapped grid $n_P(G)$, both $|pq|$ and $\angle(\eta_p, \eta_q)$ are in $O(\sqrt{\varepsilon})$. We shortcut the shortest path on ∂P from s to t to form a supported path σ that passes through each cell at most once. In this manner, we identify $O(1/\varepsilon)$ points p_1, \dots, p_m on ∂P , where p_i (resp., p_{i+1}) is the entry (resp., exit) point of the path through the i th cell in the sequence. The points p_i lie on the edges of $n_P(G)$. There are two exceptions, $p_1 = s$ and $p_m = t$, which might lie in the interior of the cell. Next, we connect each pair (p_i, p_{i+1}) by taking the shortest path on $H_{p_i} \cup H_{p_{i+1}}$. The path intersects $H_{p_i} \cap H_{p_{i+1}}$ at a point denoted q_i . (Note that q_i might be infinitesimally close to p_i .) This forms a supported path σ with $O(1/\varepsilon)$ vertices $s = p_1, q_1, p_2, q_2, \dots, q_{m-1}, p_m = t$. The only real difference from the proof in [2] is that we skip the final “trimming” step and keep the points p_i unchanged. We mention two useful, immediate consequences of Lemma 6.5.

- The folding angle at q_i is $O(\sqrt{\varepsilon})$.
- For each $1 \leq i \leq m$, the point p_i belongs to ∂P and, for $i \neq 1, m$, there exists a point $w_i = n_P(v_i)$, where $v_i \in V$, such that both $|p_i w_i|$ and $\angle(\eta_{p_i}, \eta_{w_i})$ are in $O(\lambda)$.

From σ we build a curve σ' of length $(1 + O(\varepsilon))|\sigma|$ that joins s and t outside the interior of \widehat{Q} . The classical result below shows that the shortest path on $\partial \widehat{Q}$ from s to t cannot be longer than σ' , which proves Theorem 6.3.

THEOREM 6.6 (Pogorelov [29]). *Given a convex body C , let γ be a curve joining two points $s, t \in \partial C$ outside the interior of C . Then the length of γ is at least that of the shortest path joining s and t on ∂C .*

We now explain how to construct σ' . For $0 < i < m$, let (p_i, η_{p_i}) and (q_i, η_{p_i}) be the rays emanating from p_i and q_i , respectively, in the direction normal to H_{p_i} away from P . Together with the segments $p_i q_i$ and $q_i p_{i+1}$, the four rays (p_i, η_{p_i}) , (q_i, η_{p_i}) , $(q_i, \eta_{p_{i+1}})$, and $(p_{i+1}, \eta_{p_{i+1}})$ define a polyhedral surface Σ_i , which consists of two unbounded rectangles, Σ_i^1 and Σ_i^3 , joined together at q_i by an unbounded triangle, Σ_i^2 (Figure 6). Note that the surface is in general nonplanar, but Σ_i^2 is always normal to the line $H_{p_i} \cap H_{p_{i+1}}$. Out of Σ_i we carve a polyhedral strip S_i as follows. Fix a large enough constant $c > 0$, and let K_i denote the plane $H_{p_i} + c\lambda^2 \eta_{p_i}$. In other words, K_i is a parallel copy of H_{p_i} translated by $c\lambda^2$ away from P . As usual, the superscripted K_i^+ denotes the halfspace enclosing P . Recall that w_i is the nearest neighbor of v_i defined earlier. We need to consider

$$S_i = \Sigma_i \cap \left\{ (K_i^+ \cap K_{i+1}^+) \cup (H_{w_i}^+ \cap H_{w_{i+1}}^+) \right\}.$$

Again, we have two exceptions for $i = 1, m - 1$, where we use $H_{p_1}^+$ instead of $H_{w_1}^+$ and

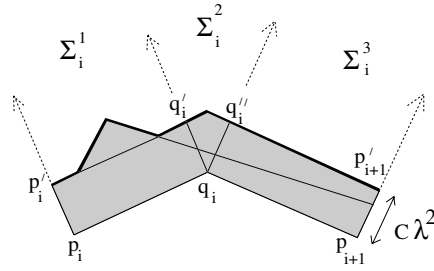


FIG. 6. The curve σ'_i .

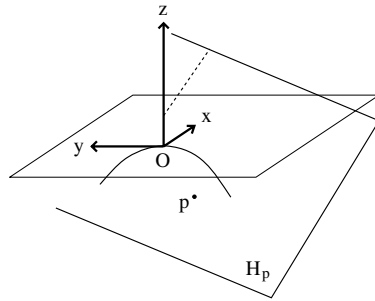


FIG. 7. How H_p intersects the xz plane.

$H_{p_m}^+$ instead of $H_{w_m}^+$.

Let $p_i p'_i$ be the edge of S_i incident to p_i collinear with η_{p_i} . We denote by σ'_i the portion of ∂S_i between p'_i and p'_{i+1} and define σ' as $\bigcup_{0 < i < m} \sigma'_i$. To provide a connection to s and t , we also add to σ' the segments $p_1 p'_1$ and $p_m p'_m$. To show that σ' is a connected curve outside the interior of \hat{Q} of length $(1 + O(\varepsilon))|\sigma|$ requires a simple technical lemma.

LEMMA 6.7 (Figure 7). *Given an orthogonal system of reference (O, xyz) , assume that P is tangent to the xy -plane at O and lies below it. Given a point p on ∂P , if $|n_P^{-1}(O)n_P^{-1}(p)| < \delta$, for some small enough $\delta > 0$, then the intersection of H_p with the xz -plane has for an equation $Z = aX + b$, where $|a| = O(\delta)$ and $0 \leq b = O(\delta^2)$.*

Proof. By Lemma 6.5, the normal to H_p forms a small angle $\theta = O(\delta)$ with the z -axis, so the plane H_p , being nonparallel to the z -axis, can be expressed as $Z = aX + cY + b$. The cross product between the normal $(a, c, -1)$ and the z -axis vector is the vector $(c, -a, 0)$. By the cross product formula, its length, which is $\sqrt{a^2 + c^2}$, is also equal to $\sqrt{a^2 + c^2 + 1} \sin \theta$. It follows that $a^2 + c^2 = O(a^2 + c^2 + 1)\delta^2$; therefore,

$$(3) \quad a^2 + c^2 = \frac{O(\delta^2)}{1 - O(\delta^2)} = O(\delta^2),$$

and hence $|a| = O(\delta)$. By convexity of P , the plane H_p intersects the nonnegative part of the z -axis, and p_z , the z -coordinate of p , is nonpositive. By (3) and $|Op| = O(\delta)$, it follows that

$$0 \leq b = p_z - ap_x - cp_y \leq \sqrt{a^2 + c^2} \sqrt{p_x^2 + p_y^2} = O(\delta^2). \quad \square$$

We examine each σ'_i separately, omitting the cases $i = 1, m - 1$, which are trivial

modifications of the general case $1 < i < m - 1$. The curve σ'_i lies outside the interior of $H_{w_i}^+ \cap H_{w_{i+1}}^+$, and hence of \widehat{Q} . It is naturally broken up into three parts, $\sigma_i^j \subset \Sigma_i^j$ ($j = 1, 2, 3$), each one of them being a polygonal curve whose edges lie in any one of four planes: K_i, K_{i+1}, H_{w_i} , and $H_{w_{i+1}}$. Applying Lemma 6.7 with $(p_i, \overrightarrow{p_i q_i}, \overrightarrow{p_i p'_i})$ in the role of (O, x, z) and w_i in the role of p , we find that H_{w_i} intersects the segment $p_i p'_i$ for c large enough (note that this c is the one in the definition of K_i); similarly, $H_{w_{i+1}}$ intersects $p_{i+1} p'_{i+1}$. This shows that p'_i is the intersection of the ray (p_i, η_{p_i}) with the plane K_i ; therefore, p'_i is *the same point* in the definition of σ'_i and σ'_{i-1} , thus proving that the curve σ' is, indeed, connected. (The danger was having p'_i defined by $H_{w_{i+1}}$.) We now bound the length of σ'_i .

- By Lemma 6.7 the slopes of the edges of σ_i^1 are chosen among 0 for K_i ; $O(\sqrt{\varepsilon})$ for K_{i+1} ; $O(\lambda)$ for H_{w_i} ; and $O(\sqrt{\varepsilon})$ for $H_{w_{i+1}}$. It follows that $|\sigma_i^1| \leq |p_i q_i| / \cos \theta$, where $\theta = O(\sqrt{\varepsilon})$; therefore, $|\sigma_i^1| = (1 + O(\varepsilon)) |p_i q_i|$. The same argument shows that $|\sigma_i^3| = (1 + O(\varepsilon)) |q_i p_{i+1}|$.
- Let q'_i, q''_i be the endpoints of the curve σ_i^2 (Figure 6), and let a, a', b, b' be the distances along the ray (q_i, η_{p_i}) from q_i to K_i, K_{i+1}, H_{w_i} , and $H_{w_{i+1}}$, respectively. By definition of S_i ,

$$|q_i q'_i| = \max \left\{ \min \{a, a'\}, \min \{b, b'\} \right\}.$$

Obviously, $a = c\lambda^2$ and, by Lemma 6.7, $b = O(\lambda |p_i q_i| + \lambda^2)$. This implies that $|q_i q'_i| = O(\lambda |p_i q_i| + \lambda^2)$ and, by the same argument,

$$|q_i q'_i| + |q_i q''_i| = O(\lambda (|p_i q_i| + |q_i p_{i+1}|) + \lambda^2).$$

Within Σ_i^2 , the curve σ_i^2 is a polygonal line consisting of at most a constant number of edges. It is not difficult to see that for any vertex v of σ_i^2 (including q'_i and q''_i), the angle between $q_i v$ and edges of σ_i^2 incident to v is $\pi/2 \pm O(\sqrt{\varepsilon})$. This follows from a simple geometric observation: given any plane H whose normal makes with $q_i v$ an angle at most α , the angle formed by $q_i v$ and any line on H lies in the range $[\pi/2 - \alpha, \pi/2 + \alpha]$. Since any of the edges of σ_i^2 lies on one of four planes, K_i, K_{i+1}, H_{w_i} , and $H_{w_{i+1}}$, and the normal of each of them makes an angle of $O(\sqrt{\varepsilon})$ with $q_i v$, the claim follows. Because the folding angle of $O(\sqrt{\varepsilon})$ can be assumed to be less than, say, $\pi/2$, this implies that the curve σ_i^2 lies entirely at a distance $O(|q_i q'_i| + |q_i q''_i|)$ from q_i . It follows that $|\sigma_i^2| = O(|q_i q'_i| + |q_i q''_i|) \sqrt{\varepsilon}$.

Putting everything together we find that

$$|\sigma'_i| = (1 + O(\varepsilon) + O(\lambda \sqrt{\varepsilon})) (|p_i q_i| + |q_i p_{i+1}|) + O(\lambda^2 \sqrt{\varepsilon}).$$

In view of the fact that $|p_1 p'_1| = |p_m p'_m| = c\lambda^2$, summing up over all $|\sigma'_i|$'s (there are $O(1/\varepsilon)$ of them),

$$\begin{aligned} |\sigma'| &= (1 + O(\varepsilon) + O(\lambda \sqrt{\varepsilon})) |\sigma| + O(\lambda^2 / \sqrt{\varepsilon}) \\ &= (1 + O(\varepsilon)) |\sigma| + O(\varepsilon) \\ &= (1 + O(\varepsilon)) |\sigma|, \end{aligned}$$

which completes the proof of Theorem 6.3. Note that the setting of λ is made to ensure that the additive term $O(\lambda^2 / \sqrt{\varepsilon})$ is $O(\varepsilon)$.

Acknowledgments. We wish to thank Pankaj Agarwal, Funda Ergun, Sariel Har-Peled, Joe Mitchell, and Ronitt Rubinfeld for several helpful discussions. We also wish to thank the anonymous referees for their comments and suggestions.

REFERENCES

- [1] P. K. AGARWAL, S. HAR-PELED, AND M. KARIA, *Computing approximate shortest paths on convex polytopes*, *Algorithmica*, 33 (1999), pp. 227–242.
- [2] P. K. AGARWAL, S. HAR-PELED, M. SHARIR, AND K. VARADARAJAN, *Approximating shortest paths on a convex polytope in three dimensions*, *J. ACM*, 44 (1997), pp. 567–584.
- [3] P. K. AGARWAL AND J. ERICKSON, *Geometric range searching and its relatives*, in *Advances in Discrete and Computational Geometry*, B. Chazelle, J. E. Goodman, and R. Pollack, eds., *Contemp. Math.* 223, AMS, Providence, RI, 1999, pp. 1–56.
- [4] G. BAREQUET AND S. HAR-PELED, *Efficiently approximating the minimum-volume bounding box of a point set in three dimensions*, *J. Algorithms*, 38 (2001), pp. 91–109.
- [5] B. CHAZELLE, *The Discrepancy Method: Randomness and Complexity*, Cambridge University Press, Cambridge, UK, 2000.
- [6] B. CHAZELLE AND D. P. DOBKIN, *Intersection of convex objects in two and three dimensions*, *J. ACM*, 34 (1987), pp. 1–27.
- [7] J. CHEN AND Y. HAN, *Shortest paths on a polyhedron*, in *Proceedings of the Sixth Annual Symposium on Computational Geometry*, ACM, New York, 1990, pp. 360–369.
- [8] K. L. CLARKSON AND P. W. SHOR, *Applications of random sampling in computational geometry*, II, *Discrete Comput. Geom.*, 4 (1989), pp. 387–421.
- [9] K. L. CLARKSON, *Las Vegas algorithms for linear and integer programming when the dimension is small*, *J. ACM*, 42 (1995), pp. 488–499.
- [10] A. CZUMAJ, F. ERGUN, L. FORTNOW, A. MAGEN, I. NEWMAN, R. RUBINFELD, AND C. SOHLER, *Sublinear-time approximation of Euclidean minimum spanning tree*, in *Proceedings of the Fourteenth Annual Symposium on Discrete Algorithms*, ACM, New York, 2003, pp. 813–822.
- [11] A. CZUMAJ AND C. SOHLER, *Property testing with geometric queries*, in *Proceedings of the 9th Annual European Symposium on Algorithms*, Springer-Verlag, Heidelberg, 2001, pp. 266–277.
- [12] A. CZUMAJ, C. SOHLER, AND M. ZIEGLER, *Property testing in computational geometry*, in *Proceedings of the 8th Annual European Symposium on Algorithms*, Springer-Verlag, Heidelberg, 2000, pp. 155–166.
- [13] M. DE BERG, M. VAN KREVELD, M. OVERMARS, AND O. SCHWARZKOPF, *Computational Geometry: Algorithms and Applications*, Springer-Verlag, Berlin, 1997.
- [14] L. DEVROYE, E. P. MÜCKE, AND B. ZHU, *A note on point location in Delaunay triangulations of random points*, *Algorithmica*, 22 (1998), pp. 477–482.
- [15] D. P. DOBKIN AND D.G. KIRKPATRICK, *Determining the separation of preprocessed polyhedra—a unified approach*, in *Automata, Languages and Programming*, Springer-Verlag, New York, 1990, pp. 400–413.
- [16] R. M. DUDLEY, *Metric entropy of some classes of sets with differentiable boundaries*, *J. Approx. Theory*, 10 (1974), pp. 227–236.
- [17] D. EPPSTEIN, *Dynamic Euclidean minimum spanning trees and extrema of binary functions*, *Discrete Comput. Geom.*, 13 (1995), pp. 111–122.
- [18] F. ERGUN, S. KANNAN, KUMAR, S. RAVI, R. RUBINFELD, AND M. VISWANATHAN, *Spot-checkers*, *J. Comput. System Sci.*, 60 (2000), pp. 717–751.
- [19] B. GÄRTNER AND E. WELZL, *A simple sampling lemma: Analysis and applications in geometric optimization*, *Discrete Comput. Geom.*, 25 (2001), pp. 569–590.
- [20] M. GRÖTSCHEL, L. LOVÁSZ, AND A. SCHRIJVER, *Geometric Algorithms and Combinatorial Optimization*, Springer-Verlag, Berlin, 1988.
- [21] S. HAR-PELED, *Approximate shortest-path and geodesic diameter on convex polytopes in three dimensions*, *Discrete Comput. Geom.*, 21 (1999), pp. 217–231.
- [22] S. KAPOOR, *Efficient computation of geodesic shortest paths*, in *Proceedings of the 31st Annual Symposium on Theory of Computing*, ACM, New York, 1999, pp. 770–779.
- [23] J. MATOŮŠEK, *Geometric range searching*, *ACM Comput. Surv.*, 26 (1994), pp. 421–461.
- [24] K. MEHLHORN, S. NAHER, T. SCHILZ, S. SCHIRRA, M. SEEL, R. SEIDEL, AND C. UHRIG, *Checking geometric programs or verification of geometric structures*, *Comput. Geom.*, 12 (1999), pp. 85–103.
- [25] J. S. B. MITCHELL, D. M. MOUNT, AND C. H. PAPADIMITRIOU, *The discrete geodesic problem*,

- SIAM J. Comput., 16 (1987), pp. 647–668.
- [26] J. S. B. MITCHELL, *An algorithmic approach to some problems in terrain navigation*, in *Autonomous Mobile Robots: Perception, Mapping and Navigation*, IEEE Computer Society Press, Los Alamitos, CA, 1991, pp. 408–427.
 - [27] E. P. MÜCKE, I. SAIAS, AND B. ZHU, *Fast randomized point location without preprocessing in two and three-dimensional Delaunay triangulations*, *Comput. Geom.*, 12 (1999), pp. 63–83.
 - [28] K. MULMULEY, *Output sensitive and dynamic constructions of higher order Voronoi diagrams and levels in arrangements*, *J. Comput. System Sci.*, 47 (1993), pp. 437–458.
 - [29] A. V. POGORELOV, *Extrinsic geometry of convex surfaces*, *Transl. Math. Monogr.* 35, AMS, Providence, RI, 1973.
 - [30] D. RON, *Property testing*, in *Handbook on Randomization*, Vol. II, S. Rajasekaran, P. M. Pardalos, J. H. Reif, and J. D. P. Rolim, eds., Kluwer Academic Publishers, Dordrecht, The Netherlands, 2001, pp. 597–649.
 - [31] R. SEIDEL, *Small-dimensional linear programming and convex hulls made easy*, *Discrete Comput. Geom.*, 6 (1991), pp. 423–434.
 - [32] M. SHARIR AND A. SCHORR, *On shortest paths in polyhedral spaces*, *SIAM J. Comput.*, 15 (1985), pp. 193–215.
 - [33] A. C. YAO, *Probabilistic computations: Towards a unified measure of complexity*, in *Proceedings of the 18th Annual Foundations of Computer Science*, IEEE Computer Society Press, Los Alamitos, CA, 1977, pp. 222–227.

LOWNESS FOR THE CLASS OF SCHNORR RANDOM REALS*

BJØRN KJOS-HANSEN[†], ANDRÉ NIES[‡], AND FRANK STEPHAN[§]

Abstract. We answer a question of Ambos-Spies and Kučera in the affirmative. They asked whether, when a real is low for Schnorr randomness, it is already low for Schnorr tests.

Key words. lowness, randomness, Schnorr randomness, Turing degrees, recursion theory, computability theory

AMS subject classifications. 68Q30, 03D25, 03D28

DOI. 10.1137/S0097539704446323

1. Introduction. In an influential 1966 paper [9], Martin-Löf proposed an algorithmic formalization of the intuitive notion of randomness for infinite sequences of 0's and 1's. His formalization was based on an effectivization of a test concept from statistics, by means of uniformly recursively enumerable (r.e.) sequences of open sets. Martin-Löf's proposal addressed some insufficiencies in an earlier algorithmic concept of randomness proposed by Church [3], who had formalized a notion now called computable stochasticity. However, Schnorr [13] criticized Martin-Löf's notion as too strong, because it was based on an r.e. test concept rather than a computable notion of tests. He suggested that one should base a formalization of randomness on computable betting strategies (also called martingales), in a way that would still overcome the problem that Church's concept was too weak. In present terminology, a real Z is computably random if no computable betting strategy succeeds along Z ; that is, for each computable betting strategy there is a finite upper bound on the capital that it reaches. The real Z is Schnorr random if no martingale succeeds *effectively*. Here effective success means that the capital at $Z \upharpoonright n$ exceeds $f(n)$ infinitely often, for some unbounded computable function f . See [1] for more on the history of these ideas.

We recall some definitions. The Cantor space 2^ω is the set of infinite binary sequences; these are called *reals* and are identified with a set of integers, i.e., subsets of ω . If $\sigma \in 2^{<\omega}$, that is, σ is a finite binary sequence, then we denote by $[\sigma]$ the set of reals that extend σ . These form a basis of clopen sets for the usual discrete topology on 2^ω . Write $|\sigma|$ for the length of $\sigma \in 2^{<\omega}$. The Lebesgue measure μ on 2^ω is defined by stipulating that $\mu[\sigma] = 2^{-|\sigma|}$. With every set $U \subseteq 2^{<\omega}$ we associate the open set $[U]^\preceq = \bigcup_{\sigma \in U} [\sigma]$. The empty sequence is denoted λ . If $\sigma, \tau \in 2^{<\omega}$ and σ is a prefix of τ , then we write $\sigma \preceq \tau$. If $\sigma \in 2^{<\omega}$ and $i \in \{0, 1\}$, then σi denotes the string of length $|\sigma| + 1$ extending σ whose final entry is i . The concatenation of two strings σ

*Received by the editors October 15, 2004; accepted for publication (in revised form) July 19, 2005; published electronically January 6, 2006. The first author was supported by a Marie Curie Fellowship of the European Community Programme "Improving Human Potential" under contract HPMF-CT-2002-01888. The second author was partially supported by University of Auckland New Staff research grant 3603229/9343, and by the Marsden fund of New Zealand, 03-UOA-130.

<http://www.siam.org/journals/sicomp/35-3/44632.html>

[†]Department of Mathematics, University of Connecticut, Storrs, CT 06269 (bjorn@math.uconn.edu).

[‡]Department of Computer Science, University of Auckland, Private Bag 92019, Auckland, New Zealand (andre@cs.auckland.ac.nz).

[§]School of Computing and Department of Mathematics, National University of Singapore, 3 Science Drive 2, Singapore 117543, Republic of Singapore (fstefhan@comp.nus.edu.sg).

and τ is denoted $\sigma\tau$. The empty set is denoted \emptyset , and inclusion of sets is denoted by \subseteq . If A is a real and $n \in \omega$, then $A \upharpoonright n$ is the prefix of A consisting of the first n bits of A . Letting $A(n)$ denote bit n of A , we have $A \upharpoonright n = A(0)A(1) \cdots A(n-1)$.

Given $\alpha \in 2^{<\omega}$ and a measurable set $C \subseteq 2^\omega$, we let $\mu_\alpha C = \frac{\mu(C \cap [\alpha])}{\mu[\alpha]}$. For an open set W we let

$$W|\sigma = \bigcup \{[\tau] : \tau \in 2^{<\omega}, [\sigma\tau] \subseteq W\}.$$

Note in particular that $\mu_\sigma W = \mu(W|\sigma)$ and $\mu_\lambda W = \mu W$.

Fixing some effective correspondence between the set of finite subsets of ω and ω , we let D_e be the e th finite subset of ω under this correspondence. In other words, e is a strong, or canonical, index for the finite set D_e . Similarly, we let S_e be the e th finite subset of $2^{<\omega}$ under a suitable correspondence. Thus S_e is a finite set of strings, and $[S_e]^\preceq = \bigcup_{\sigma \in S_e} [\sigma]$ is then the clopen set coded by $e \in \omega$. We use the Cantor pairing function, namely the bijection $p : \omega^2 \rightarrow \omega$ given by $p(n, s) = \frac{(n+s)^2 + 3n + s}{2}$, and write $\langle n, s \rangle = p(n, s)$.

A *Martin-Löf test* is a set $U \subseteq \omega \times 2^\omega$ such that $\mu U_n \leq 2^{-n}$, where U_n denotes the n th section of U , and U_n is a Σ_1^0 class, uniformly in n . If, in addition, μU_n is a computable real, uniformly in n , then U is called a *Schnorr test*. Z is *Martin-Löf random* if for each Martin-Löf test U there is an n such that $Z \notin U_n$, and *Schnorr random* if for each Schnorr test U there is an n such that $Z \notin U_n$. The notion of Schnorr randomness is unchanged if we instead define a Schnorr test to be a Martin-Löf test for which $\mu U_n = 2^{-n}$ for each $n \in \omega$.

Concepts encountered in computability theory are usually based on some notion of computation, and therefore have relativized forms. For instance, we may relativize the tests and randomness notions above to an oracle A . If $\mathcal{C} = \{X : X \text{ is Martin-Löf random}\}$, then the relativization is $\mathcal{C}^A = \{X : X \text{ is Martin-Löf random relative to } A\}$ (meaning that Σ_1^0 classes are replaced by $\Sigma_1^{0,A}$ classes). In general, if \mathcal{C} is such a relativizable class, we say that A is *low for \mathcal{C}* if $\mathcal{C}^A = \mathcal{C}$. If \mathcal{C} is a randomness notion, more computational power means a smaller class, namely $\mathcal{C}^A \subseteq \mathcal{C}$ for any A . Being low for \mathcal{C} means having small computational power (in a sense that depends on \mathcal{C}). In particular, the low-for- \mathcal{C} reals are closed downward under Turing reducibility.

The randomness notions for which lowness was first considered are Martin-Löf and Schnorr randomness. Kučera and Terwijn [8] constructed a noncomputable r.e. set of integers A which is low for Martin-Löf randomness, answering a question of Zambella [16]. In the paper [14] it is shown that there are continuously many reals that are low for Schnorr randomness.

An important difference between the two randomness notions is that for Martin-Löf randomness, but not for Schnorr randomness, there is a universal test R . Thus, Z is not Martin-Löf random iff $Z \in \bigcap_{b \in \omega} R_b$. Therefore, in the Schnorr case, an apparently stronger lowness notion is being *low for Schnorr tests*, or *S_0 -low* in the terminology of [1]: A is low for Schnorr tests if for each Schnorr test U^A relative to A there is an unrelativized Schnorr test V such that $\bigcap_n U_n^A \subseteq \bigcap_n V_n$. This implies that A is low for Schnorr randomness, or *S -low* in the terminology of [1]. Ambos-Spies and Kučera asked if the two notions coincide. We answer this question in the affirmative.

Terwijn and Zambella [14] actually constructed oracles A that are low for Schnorr tests. They first gave a characterization of this lowness property via a notion of traceability, a restriction on the possible sequence of values of the functions computable from A . They showed that A is low for Schnorr tests iff A is computably traceable

(see formal definition in the next section). Then they constructed continuously many computably traceable reals. We answer the question of Ambos-Spies and Kučera by showing that each real which is low for Schnorr randomness is in fact computably traceable.

Towards this end, it turns out to be helpful to have a more general view of lowness. We consider lowness for any pair of randomness notions \mathcal{C}, \mathcal{D} with $\mathcal{C} \subseteq \mathcal{D}$.

DEFINITION 1.1. *A is in $\text{Low}(\mathcal{C}, \mathcal{D})$ if $\mathcal{C} \subseteq \mathcal{D}^A$. We write $\text{Low}(\mathcal{C})$ for $\text{Low}(\mathcal{C}, \mathcal{C})$.*

Clearly, if $\mathcal{C} \subseteq \tilde{\mathcal{C}} \subseteq \tilde{\mathcal{D}} \subseteq \mathcal{D}$ are randomness notions, and the inclusions relativize (so $\tilde{\mathcal{D}}^A \subseteq \mathcal{D}^A$ for each real A), then $\text{Low}(\tilde{\mathcal{C}}, \tilde{\mathcal{D}}) \subseteq \text{Low}(\mathcal{C}, \mathcal{D})$. That is, we make the class $\text{Low}(\tilde{\mathcal{C}}, \tilde{\mathcal{D}})$ larger by decreasing \mathcal{C} or increasing \mathcal{D} . Let MR, CR, and SR denote the classes of Martin-Löf random, computably random (defined below), and Schnorr random reals, respectively. Thus, for instance, $\text{Low}(\text{MR}, \text{CR})$ is the class of oracles A such that each Martin-Löf random real is computably random in A . We will characterize lowness for any pair of randomness notions $\mathcal{C} \subseteq \mathcal{D}$ with $\mathcal{C}, \mathcal{D} \in \{\text{MR}, \text{CR}, \text{SR}\}$.

Recall that Ω denotes the halting probability of a universal prefix machine. Ω is a Martin-Löf random *r.e. real*, i.e., a real that can be effectively approximated from below. Given $\mathcal{D} \supseteq \text{MR}$, an interesting lowness notion obtained by weakening $\text{Low}(\text{MR}, \mathcal{D})$ is $\text{Low}(\{\Omega\}, \mathcal{D})$. That is, instead of $\text{MR} \subseteq \mathcal{D}^A$ one merely requires that $\Omega \in \mathcal{D}^A$. We denote this class by $\text{Low}(\Omega, \mathcal{D})$. In [12], the case $\mathcal{D} = \text{MR}$ is studied. The authors show that the class coincides with $\text{Low}(\text{MR})$ on the Δ_2^0 reals but not in general. In fact, a Martin-Löf random real is 2-random iff it is in $\text{Low}(\Omega, \text{MR})$.

Here we investigate the class $\text{Low}(\Omega, \text{SR})$. We show that A is $\text{Low}(\text{MR}, \text{SR})$ iff A is r.e. traceable. Moreover, the weaker assumption $\Omega \in \text{SR}^A$ still implies that A is array computable (there is a function $f \leq_{\text{wtt}} \emptyset'$ bounding all functions computable from A , on almost all inputs). Thus for r.e. sets of integers A , A being $\text{Low}(\text{MR}, \text{SR})$ is in fact equivalent to $\Omega \in \text{SR}^A$ by Ishmukhametov [5]. We also provide an example of a real A which is array computable but not $\text{Low}(\Omega, \text{SR})$.

2. Main concepts.

2.1. Martingales. For our purposes, a *martingale* is a function $M : 2^{<\omega} \mapsto \mathbb{Q}$ (where \mathbb{Q} is the set of rational numbers) such that (i) the domain of M is $2^{<\omega}$, or $2^{\leq n} = \{\sigma \in 2^{<\omega} : |\sigma| \leq n\}$ for some n , (ii) $M(\lambda) \leq 1$, and (iii) M has the martingale property $M(x0) + M(x1) = 2M(x)$ whenever the strings $x0, x1$ belong to the domain of M . A martingale M *succeeds* on a sequence $Z \in 2^\omega$ if

$$\limsup_{n \rightarrow \infty} M(Z \upharpoonright n) = \infty.$$

A real is *computably random* if no computable martingale succeeds on it.

A martingale M *effectively succeeds* on a sequence Z if there is a nondecreasing and unbounded computable function $h : \omega \rightarrow \omega$ such that

$$\limsup_{n \rightarrow \infty} M(Z \upharpoonright n) - h(n) > 0.$$

Equivalently (since we are considering integer-valued functions), $\exists^\infty n M(Z \upharpoonright n) > h(n)$. We can now state the characterization of Schnorr randomness in terms of martingales: a real Z is Schnorr random iff no computable martingale effectively succeeds on Z .

2.2. Traceability. Let W_e denote the e th r.e. set of integers in some standard list. A real A is *r.e. traceable* if there is a computable function p , called a *bound*, such that for every $f \leq_T A$ there is a computable function r such that for all x we have $|W_{r(x)}| \leq p(x)$ and $f(x) \in W_{r(x)}$.

The following is a stronger notion than r.e. traceability. A is *computably traceable* if there is a computable p such that for every $f \leq_T A$ there is a computable r such that for all x we have $|D_{r(x)}| \leq p(x)$ and $f(x) \in D_{r(x)}$.

It is interesting to notice that it does not matter what bound p one chooses as a witness for traceability; see the following.

PROPOSITION 2.1 (see Terwijn and Zambella [14]). *Let A be a real that is computably traceable with bound p . Then for any monotone and unbounded computable function p' , A is computably traceable with bound p' . The same holds for r.e. traceability.*

The result of Terwijn and Zambella is the following.

THEOREM 2.2 (see [14]). *A real A is low for Schnorr tests iff A is computably traceable.*

3. Statement of the main result.

THEOREM 3.1.

(I) *A is Low(MR, SR) iff A is r.e. traceable.*

(II) *A is Low(CR, SR) iff A is Low(SR) iff A is computably traceable.*

We make some remarks about the proofs and fill in the details in the next section. We obtain Theorem 3.1(I) by modifying the methods in [14] to the case of r.e. traces instead of computable ones.

As for Theorem 3.1(II), by Theorem 2.2 if A is computably traceable, then A is low for Schnorr tests. Hence A is certainly Low(SR), and therefore also Low(CR, SR). It remains only to show that each real $A \in \text{Low}(\text{CR}, \text{SR})$ is computably traceable. To see that this is so, take the following three steps:

1. Recall that A is *hyperimmune-free* if for each $g \leq_T A$ there is a computable f such that for all x we have $g(x) \leq f(x)$. As a first step towards proving Theorem 3.1(II), Bedregal and Nies [2] showed that each $A \in \text{Low}(\text{CR}, \text{SR})$ is hyperimmune-free (see Lemma 4.9 below). To see this, assume that A is not, so there is a function $g \leq_T A$ not dominated by any computable function f . Define a martingale $L \leq_T A$ which succeeds in the sense of Schnorr, with the computable lower bound $h(n) = n/4$, on some $Z \in \text{CR}$. One uses here that g is infinitely often above the running time of each computable martingale. (Special care has to be taken with the partial martingales, which results in a real Z that is only Δ_3^0 .)

2. If A is hyperimmune-free and r.e. traceable, then A is computably traceable. If we let $g \leq_T A$, then the first stage where $g(x)$ appears in a given trace for g can be computed relative to A .

3. Now each A in Low(CR, SR) is r.e. traceable by Theorem 3.1(I), and hence by the above is computably traceable, and Theorem 3.1(II) follows.

We discuss lowness for the remaining pairs of randomness notions. Nies has shown that A is Low(MR, CR) iff A is Low(MR) iff A is K -trivial, where A is K -trivial if for all n $K(X \upharpoonright n) \leq K(n) + O(1)$ (see [11]). Here $K(\sigma)$ denotes the prefix-free Kolmogorov complexity of $\sigma \in 2^{<\omega}$. Finally, he shows that a real A which is Low(CR) is computable; namely, A is both K -trivial and hyperimmune-free. Since all K -trivial reals are Δ_2^0 , and all hyperimmune-free Δ_2^0 reals are computable, the conclusion follows.

4. Proof of the main result. We first need to develop a few useful facts from measure theory.

DEFINITION 4.1. *A measurable set A has density d at a real X if*

$$\lim_{n \rightarrow \infty} \mu_{(X|n)}A = d.$$

A basic result is the following.

THEOREM 4.2 (Lebesgue density theorem). *Let $\Xi(A) = \{X : A \text{ has density } 1 \text{ at } X\}$. If A is a measurable set, then so is $\Xi(A)$, and the measure of the symmetric difference of A and $\Xi(A)$ is zero.*

COROLLARY 4.3. *Let C be a measurable subset of 2^ω , with $\mu C > 0$. Then for each $\delta < 1$ there is an $\alpha \in 2^{<\omega}$ such that $\mu_\alpha C \geq \delta$.*

We will use the following consequence of Corollary 4.3.

LEMMA 4.4. *Let $0 < \epsilon \leq 1$. If $U_n, n \in \omega$, and V are open subsets of 2^ω with $\bigcap_{n \in \omega} U_n \subseteq V$ and $\mu V < \epsilon$, then there exist σ and n such that $\mu_\sigma(U_n - V) = 0$ and $\mu_\sigma V < \epsilon$.*

Proof. Suppose otherwise; we shall obtain a contradiction by constructing a real in $\bigcap_{n \in \omega} U_n - V$. Let $\sigma_0 = \lambda$ and assume we have defined σ_n such that $\mu_{\sigma_n} V < \epsilon$. By hypothesis, $\mu_{\sigma_n}(U_n - V) > 0$, and thus there is a $[\tau] \subseteq U_n$ such that $\mu_{\sigma_n}([\tau] - V) > 0$. In particular, $\tau \succeq \sigma_n$ and $\mu_\tau V < 1$. Let $C = 2^\omega - V$, a closed and hence measurable set. By Corollary 4.3 applied to C (and with 2^ω replaced by $[\tau]$), there exists $\sigma_{n+1} \succeq \tau$ such that $\mu_{\sigma_{n+1}} V < \epsilon$. Let X be the real that extends all σ_n 's constructed in this way. Since $[\sigma_{n+1}] \subseteq U_n$ for all n , we have that $X \in \bigcap_{n \in \omega} U_n$. However, $[\sigma_n] \not\subseteq V$ for every n , so, since V is open, $X \notin V$. This contradiction completes the proof. \square

We now get to the proof of Theorem 3.1. First we show Theorem 3.1(I), namely, that A is Low(MR, SR) iff A is r.e. traceable. We start with the “ \Leftarrow ” direction.

LEMMA 4.5. *If A is r.e. traceable, then A is Low(MR, SR).*

Proof. Assume that A is r.e. traceable and that U^A is a Schnorr test relative to A . Let $U_{n,s}^A, n, s \in \omega$, be clopen sets, $U_{n,s}^A \subseteq U_{n,s+1}^A, U_n^A = \bigcup_{s \in \omega} U_{n,s}^A$, such that the $U_{n,s}^A$ are $\Delta_1^{0,A}$ classes uniformly in n and s . As $\mu U_n^A = 2^{-n}$, we may assume that $\mu U_{n,s}^A > 2^{-n}(1 - 2^{-s})$. Let f be an A -computable function such that $[S_{f(\langle n,s \rangle)}]^\preceq = U_{n,s}^A$. Since A is r.e. traceable and $f \leq_T A$, we can let T be an r.e. trace of f . By Proposition 2.1, we may choose T such that in addition $|T_x| \leq x$ for each $x > 0$.

We now want to define a subtrace \hat{T} of T , i.e., $\hat{T}_{\langle n,s \rangle} \subseteq T_{\langle n,s \rangle}$ for each n, s . The intent is that the open sets defined via \hat{T} are small enough to give us a Martin-Löf test containing $\bigcap_{n \in \omega} U_n^A$, and nothing important is in $T_{\langle n,s \rangle} - \hat{T}_{\langle n,s \rangle}$. Thus let $\hat{T}_{\langle n,s \rangle}$ be the set of $e \in T_{\langle n,s \rangle}$ such that $2^{-n}(1 - 2^{-s}) \leq \mu[S_e]^\preceq \leq 2^{-n}$ and $[S_e]^\preceq \supseteq [S_d]^\preceq$ for some $d \in \hat{T}_{\langle n,s-1 \rangle}$, where $\hat{T}_{\langle n,-1 \rangle} = \omega$. Let

$$V_n = \bigcup \left\{ [S_e]^\preceq : e \in \hat{T}_{\langle n,s \rangle}, s \in \omega \right\}.$$

Then $\mu V_n \leq 2^{-n}|\hat{T}_{\langle n,0 \rangle}| + \sum_{s \in \omega} 2^{-s}2^{-n}|\hat{T}_{\langle n,s \rangle}|$. Since $|\hat{T}_{\langle n,s \rangle}| \leq |T_{\langle n,s \rangle}| \leq \langle n, s \rangle$ for $\langle n, s \rangle \neq 0$, and $\langle n, s \rangle$ has only polynomial growth in n and s , it is clear that $\sum_{s \in \omega} 2^{-s}2^{-n}|\hat{T}_{\langle n,s \rangle}|$ is finite and goes effectively to 0 as $n \rightarrow \infty$; hence the same can be said of μV_n . Thus there is a recursive function f such that $\mu V_{f(n)} \leq 2^{-n}$. Let $\tilde{V}_n = V_{f(n)}$. Then \tilde{V} is a Martin-Löf test and $\bigcap_n U_n^A \subseteq \bigcap_n \tilde{V}_n$. That is, each Schnorr test relative to A is contained in a Martin-Löf test. It follows that each real that is Martin-Löf random is Schnorr random relative to A , and the proof is complete. \square

Next we will show the “ \Rightarrow ” direction of Theorem 3.1(I). The proof is similar to the “ \Rightarrow ” of Theorem 2.2.

DEFINITION 4.6. For $k, l \in \omega$ define the clopen set

$$B_{k,l} = \bigcup \{[\tau 1^k] : \tau \in 2^{<\omega}, |\tau| = l\},$$

where 1^k is a string of 1’s of length k .

Note that $\mu B_{k,l} = 2^{-k}$ for all l .

LEMMA 4.7. If $A \in 2^\omega$ is Low(MR, SR), then A is r.e. traceable.

Proof. Note that A is Low(MR, SR) iff for every Schnorr test U^A relative to A , $\bigcap_{n \in \omega} U_n^A \subseteq \bigcap_{b \in \omega} R_b$ (recall that R is a universal Martin-Löf test).

Oversimplifying a bit, one can say that the proof below goes as follows. We code a given $g \leq_T A$ into a Schnorr test U^g relative to A . Then, by hypothesis, $\bigcap_n U_n^g \subseteq \bigcap_n R_n$; in fact we will use only the fact that $\bigcap_n U_n^g \subseteq R_3$. We then define an r.e.trace T ; namely, T_k is the set of l such that $B_{k,l} - R_3$ has small measure in some sense. Since R_3 has rather small measure, $B_{k,l} - R_3$ will tend to have big measure, which means that there will be only a few l for which $B_{k,l} - R_3$ has small measure; in other words, T_k has small size. Moreover, we make sure T is a trace for g so that A is r.e. traceable.

We now give the proof details. Suppose that we want to find a trace for a given function $g \leq_T A$. We define the test U^g by stipulating that

$$U_n^g = \bigcup_{k > n} B_{k,g(k)}.$$

It is easy to see that μU_n^g can be approximated computably in A , so after taking a subsequence of U_n^g , $n \in \omega$, we may assume that U^g is a Schnorr test relative to A . Hence, by assumption, $\bigcap U^g \subseteq \bigcap_{b \in \omega} R_b$. Thus $V = R_3$ contains $\bigcap U^g$ and $\mu V < \frac{1}{4}$. We may assume throughout that $g(k) \geq k$ for every k because from a trace for $g(k) + k$ one can obtain a trace for g with the same bound. By Lemma 4.4, there exist σ and n such that $\mu_\sigma(U_n^g - V) = 0$ and $\mu_\sigma V < 1/4$. As $U_0^g \supseteq U_1^g \supseteq \dots$, we can choose σ and n with the additional property $n \geq |\sigma|$. Hence for each $k > n$, we have $g(k) \geq k > n \geq |\sigma|$ and hence $g(k) \geq |\sigma|$.

Let $\tilde{V} = V|\sigma$, let $\tilde{g}(k) = \max\{0, g(k) - |\sigma|\}$, and take

$$T_k = \left\{ l : \mu(B_{k,l} - \tilde{V}) < 2^{-(l+3)} \right\}.$$

Note that for each $l \in \omega$, if $l \geq |\sigma|$, then $B_{k,l}|\sigma = B_{k,l-|\sigma|}$. Thus, since $g(k) \geq |\sigma|$,

$$U_n^g|\sigma = \bigcup_{k > n} B_{k,g(k)}|\sigma = \bigcup_{k > n} B_{k,g(k)-|\sigma|} = U_n^{\tilde{g}},$$

and so $\mu(U_n^{\tilde{g}} - \tilde{V}) = \mu_\sigma(U_n^g - V) = 0$. Hence $\tilde{g}(k) \in T_k$ for all $k > n$.

Since \tilde{V} is a Σ_1^0 class, it is evident that T is an r.e. set of integers; indeed $B_{k,l} - \tilde{V}$ is a Π_1^0 class, and thus we can enumerate the fact that certain basic open sets $[\sigma]$ are disjoint from it, until the measure remaining is as small as required. A trace for g is obtained as follows:

$$G_k = \begin{cases} \{l + |\sigma| : l \in T_k\} & \text{if } k > n, \\ \{g(k)\} & \text{if } k \leq n. \end{cases}$$

We now show that G is a trace for g ; i.e., for all $k \in \omega$, $g(k) \in G_k$. If $k \leq n$, then this holds by definition of G_k ; thus suppose $k > n$. Then $g(k) > k > n > |\sigma|$, so $\tilde{g}(k) = g(k) - |\sigma|$, so $g(k) = \tilde{g}(k) + |\sigma|$. As $k > n$, $\tilde{g}(k) \in T_k$ and hence $g(k) \in G_k$.

Clearly G is r.e.; thus it remains to show that $|G_k|$ is computably bounded, independently of g . As $|G_k| = |T_k|$ for $k > n$ and $|G_k| = 1$ for $k \leq n$, this is a consequence of Lemma 4.8 below. \square

LEMMA 4.8. *If \tilde{V} is a measurable set with $\mu\tilde{V} < \frac{1}{4}$, and $T_k = \{l : \mu(B_{k,l} - \tilde{V}) < 2^{-(l+3)}\}$, then for $k \geq 1$, $|T_k| < 2^k k$.*

Proof. Observe that, by definition of T_k ,

$$\sum_{l \in T_k} \mu(B_{k,l} - \tilde{V}) < \sum_{l \in T_k} 2^{-(l+3)} \leq \frac{1}{8} \sum_{l \in \omega} 2^{-l} = \frac{1}{4},$$

so

$$\mu \bigcup_{l \in T_k} B_{k,l} - \mu\tilde{V} \leq \mu \bigcup_{l \in T_k} (B_{k,l} - \tilde{V}) \leq \frac{1}{4}.$$

As $\mu\tilde{V} < \frac{1}{4}$, we obtain that

$$\mu \bigcup_{l \in T_k} B_{k,l} < \frac{1}{2}.$$

As observed above, $\mu B_{k,l} = 2^{-k}$. Moreover, for k fixed, the $B_{k,l}$'s are mutually independent as soon as the l 's are taken sufficiently far apart. In fact, sufficiently far here means a distance of k . So for $k \geq 1$ we let T_k^* be a subset of T_k consisting of $\lfloor \frac{|T_k|}{k} \rfloor$ elements, all of which are sufficiently far apart. (Here $\lfloor a \rfloor$ is the greatest integer $\leq a$.) To show that such a set exists we may assume we are in the worst case, where the elements of T_k are closest together: say, $T_k = \{0, \dots, |T_k| - 1\}$. Then let $T_k^* = \{mk : 0 \leq m \leq \lfloor \frac{|T_k|}{k} \rfloor - 1\}$. As $(\lfloor \frac{|T_k|}{k} \rfloor - 1)k \leq |T_k| - k \leq |T_k| - 1 \in T_k$, this makes $T_k^* \subseteq T_k$. Write $\alpha = \lfloor \frac{|T_k|}{k} \rfloor$. We now have

$$\mu \bigcap_{l \in T_k} (2^\omega - B_{k,l}) \leq \mu \bigcap_{l \in T_k^*} (2^\omega - B_{k,l}) = (1 - 2^{-k})^\alpha$$

and hence

$$\begin{aligned} 1 - (1 - 2^{-k})^\alpha &\leq 1 - \mu \bigcap_{l \in T_k} (2^\omega - B_{k,l}) = \mu 2^\omega - \mu \bigcap_{l \in T_k} (2^\omega - B_{k,l}) \\ &\leq \mu \left(2^\omega - \bigcap_{l \in T_k} (2^\omega - B_{k,l}) \right) = \mu \bigcup_{l \in T_k} B_{k,l} < \frac{1}{2}. \end{aligned}$$

From the inequality above we obtain, letting $m = 2^k - 1$,

$$\left(1 - \frac{1}{m+1} \right)^\alpha = (1 - 2^{-k})^\alpha > \frac{1}{2}$$

or $\left(\frac{m+1}{m}\right)^\alpha < 2$. Now suppose $\alpha \geq m$. Then $\left(\frac{m+1}{m}\right)^\alpha \geq \left(\frac{m+1}{m}\right)^m \geq 2$ as $(m+1)^m \geq m^m + m^{m-1} \binom{m}{1} = 2m^m$. Thus we conclude $\alpha < m = 2^k - 1$. Now, by the definition of α , we have $\frac{|T_k|}{k} \leq \alpha + 1 < 2^k$ and so $|T_k| < 2^k k$; this completes the proof. \square

In order to prove Theorem 3.1(II), recall that, by Theorem 2.2, each computably traceable real is Low(SR). Thus it suffices to show that each Low(CR, SR) real is computably traceable. The first ingredient for showing this is the following result from [2].

LEMMA 4.9. *If A is Low(CR, SR), then A is hyperimmune-free.*

Proof. Suppose A is not hyperimmune-free, so that there is a function $g \leq_T A$ not dominated by any computable function. Thus, for each computable f , $\exists^\infty x f(x) \leq g(x)$. We will define a computably random real X and an A -computable martingale L that succeeds on X in the sense of Schnorr, so that A is not Low(CR, SR). In the following, α, β, γ denote finite subsets of ω , and $n_\alpha = \sum_{i \in \alpha} 2^i$ (here $n_\emptyset = 0$).

Let $\{M_e\}_{e \in \omega}$ be an effective listing of all partial computable martingales with range included in $[1/2, \infty)$. At stage t , we have a finite portion $M_e[t]$, whose domain is a subset of some set of the form $2^{\leq n}$ for some n . If X is not computably random, then $\lim_{n \rightarrow \infty} M_e(X \upharpoonright n) = \infty$ for some total M_e by [13]. Let

$$TMG = \{e : M_e \text{ is total}\}.$$

For finite sets α, β , let us in this proof say that α is a *strong subset* of β (denoted $\alpha \subseteq^+ \beta$) if $\alpha \subseteq \beta$ and moreover for each $i \in \omega$, if $i \in \beta - \alpha$, then $i > \max(\alpha)$. Thus the possibility that β contains an element smaller than some element of α is ruled out.

For certain α , and all those included in TMG , we will define strings x_α in such a way that $\alpha \subseteq^+ \beta \Rightarrow x_\alpha \preceq x_\beta$. We choose the strings in such a way that $M_e(x_\alpha)$ is bounded by a fixed constant (depending on e) for each total M_e and each α containing e . Then the set of integers

$$X = \bigcup_{e \in \omega} x_{TMG \cap [0, e]}$$

is a computably random real. On the other hand, we are able to define an A -computable martingale L which Schnorr succeeds on X . We give an inductive definition of the strings x_α , “scaling factors” $p_\alpha \in \mathbb{Q}^+$ (positive rationals) (we do not define p_\emptyset), and partial computable martingales M_α such that if x_α is defined, then

$$(1) \quad M_\alpha(x_\alpha) \text{ converges in } g(|x_\alpha|) \text{ steps and } M_\alpha(x_\alpha) < 2.$$

It will be clear that A can decide if $y = x_\alpha$, given inputs y and α .

Let $x_\emptyset = \lambda$, and let M_\emptyset be the constant zero function. (We may assume that g is such that $M_\emptyset(\lambda)$ converges in $g(0)$ steps.) Now suppose $\alpha = \beta \cup \{e\}$, where $e > \max(\beta)$, and inductively suppose that (1) holds for β . Let

$$p_\alpha = \frac{1}{2} 2^{-|x_\beta|} (2 - M_\beta(x_\beta)),$$

and let $M_\alpha = M_\beta + p_\alpha M_e$. Since M_e is a martingale on its domain, $M_e(z) \leq 2^{|z|}$ for any z . So, writing $b = M_\beta(x_\beta)$, we have $M_\alpha(x_\beta) = b + p_\alpha M_e(x_\beta) < b + p_\alpha 2^{|x_\beta|} = b + \frac{1}{2}(2 - b) = 1 + \frac{b}{2} < 1 + \frac{2}{2} = 2$ if $M_\alpha(x_\beta)$ is defined.

To define x_α , we look for a sufficiently long $x \succeq x_\beta$ such that M_α does not increase from x_β to x and $M_\alpha(x)$ converges in $g(|x|)$ steps. In detail, for larger and larger $m > |x_\beta|$, $m \geq 4n_\alpha$, if no string y , $|y| < m$, has been designated to be x_α as yet, and if $M_\alpha(z)$ (i.e., each $M_e(z), e \in \alpha$) converges in $g(m)$ steps, for each string z

of length $\leq m$, then choose x_α of length m , $x_\beta \prec x_\alpha$ such that M_α does not increase anywhere from x_β to x_α .

CLAIM 4.10. *If $\alpha \subseteq TMG$, then x_α and p_α (the latter only if $\alpha \neq \emptyset$) are defined.*

Proof. The claim is trivial for $\alpha = \emptyset$. Suppose that it holds for β , and $\alpha = \beta \cup \{e\} \subseteq TMG$, where $e > \max(\beta)$. Since the function

$$f(m) = \mu s \quad \forall e \in \alpha, \forall x \quad [|x| \leq m \Rightarrow M_e(x) \text{ converges in } s \text{ steps}]$$

is computable, there is a least $m \geq 4n_\alpha$, $m > |x_\beta|$ such that $g(m) \geq f(m)$. Since there is a path down the tree starting at x_β , where M_α does not increase, the choice of x_α can be made. \square

CLAIM 4.11. *If $\beta \subseteq^+ \alpha$ are finite sets, then $M_\beta(x) \leq M_\alpha(x)$ for all x .*

Proof. This is clear by induction from the case $\alpha = \beta \cup \{e\}$, i.e., the case where $\alpha - \beta$ has only one element. \square

CLAIM 4.12. *X is computably random.*

Proof. Suppose that M_e is total. Let $\alpha = TMG \cap [0, e]$. Suppose $\alpha \subseteq \gamma$, $\gamma' = \gamma \cup \{i\}$, $\max(\gamma) < i$, and $\gamma' \subseteq TMG$. Then $\alpha \subseteq^+ \gamma \subseteq^+ \gamma'$. Hence by Claim 4.11, for each x with $x_\gamma \preceq x \preceq x_{\gamma'}$, we have

$$p_\alpha M_e(x) \leq M_\alpha(x) \leq M_{\gamma'}(x) \leq M_{\gamma'}(x_\gamma) < 2,$$

and hence $M_e(x) < 2/p_\alpha$ for each $x \prec X$, and so the capital of M_e on X is bounded. \square

CLAIM 4.13. *There is a martingale $L \leq_T A$ which effectively succeeds on X . In fact,*

$$\exists^\infty x \prec X \quad L(x) \geq \left\lfloor \frac{|x|}{4} \right\rfloor.$$

Proof. For a string z , let $r(z) = \lfloor |z|/2 \rfloor$. We let $L = \sum_\alpha L_\alpha$, where L_α is a martingale with initial capital $L_\alpha(\lambda) = 2^{-n_\alpha}$, which bets everything along x_α from $x_\alpha \upharpoonright r(x_\alpha)$ on. More precisely, if x_α is undefined, then L_α is constant with value 2^{-n_α} . Otherwise, for convenience we let $x = x_\alpha \upharpoonright 2r(x_\alpha)$ and work with x instead of x_α ; we define L_α on a string y as follows:

- If y does not contain “half of x ,” i.e., if $x \upharpoonright r(x) \not\preceq y$, then just let $L_\alpha(y) = 2^{-n_\alpha}$.
- If y does contain “half of x ” but y and x are incompatible, then let $L_\alpha(y) = 0$.
- If y contains “half of x ” and x and y are compatible, then let $L_\alpha(y) = 2^{-n_\alpha} 2^{\min(|y|-r(x), r(x))}$.

Thus if y contains x , then $L_\alpha(y) = 2^{r(x)-n_\alpha}$, so we make no more bets once we extend x_α , and if x contains y , then $L_\alpha(y) = 2^{|y|-r(x)-n_\alpha}$; i.e., we double the capital for each correct bit of x beyond $x \upharpoonright r(x)$.

Note that $L(\lambda) = \sum_\alpha 2^{-n_\alpha}$, and, as each $k \in \omega$ has a unique binary expansion and hence is equal to n_α for a unique finite set α , we have $L(\lambda) = \sum_{k \in \omega} 2^{-k} = 2$. Moreover, it is clear that each L_α satisfies the martingale property $L_\alpha(x0) + L_\alpha(x1) = 2L_\alpha(x)$, and hence so does L .

L effectively succeeds on X . Indeed, as $|x_\alpha| \geq 4n_\alpha$, we have $L_\alpha(x_\alpha) = 2^{r(x_\alpha)-n_\alpha} \geq 2^{\lfloor |x_\alpha|/2 \rfloor - \lfloor |x_\alpha|/4 \rfloor} \geq 2^{\lfloor |x_\alpha|/4 \rfloor} \geq \lfloor |x_\alpha|/4 \rfloor$ since $2^q \geq q$ for each $q \in \omega$.

Finally, we show that $L \leq_T A$. Given input y , we use g to see if some string x , $|x| \leq 2|y|$, is x_α . If not, $L_\alpha(y) = 2^{-n_\alpha}$. Else we determine $L_\alpha(y)$ from x using the definition of L_α . \square

The second ingredient to the proof of Theorem 3.1(II) is the following fact of independent interest.

PROPOSITION 4.14. *If A is hyperimmune-free and r.e. traceable, then A is computably traceable.*

Proof. Let $f \leq_T A$, and let h be as in the definition of r.e. traceability. Let $g(x) = \mu s(f(x) \in W_{h(x),s})$ (where $W_{e,s}$ is the approximation at stage s to the r.e. set W_e). Then $g \leq_T A$, and so since A is hyperimmune-free, g is dominated by a computable function r . Thus if we replace $W_{h(x)}$ by $W_{h(x),r(x)}$, we obtain a computable trace for f . \square

Lemma 4.9 and Proposition 4.14 together establish Theorem 3.1(II): if A is Low(CR,SR), then A is r.e. traceable by Theorem 3.1(I), and hyperimmune-free by Lemma 4.9. Thus by Proposition 4.14, A is computably traceable.

As a corollary, we obtain an answer to the question of Ambos-Spies and Kučera.

COROLLARY 4.15. *A real A is S -low iff it is S_0 -low.*

Proof. This follows by Theorem 2.2 and Theorem 3.1(II), since each computably traceable real is S_0 -low. \square

5. Lowness notions related to Chaitin's halting probability. Recall that A is array computable if there is a function $f \leq_{wtt} \emptyset'$ bounding all functions computable from A on almost all inputs.

THEOREM 5.1. *If $\Omega \in \text{SR}^A$, then A is array computable.*

Proof. We show that the function $\beta(x) = \mu s \Omega_s \upharpoonright 3x = \Omega \upharpoonright 3x$ dominates each function $\alpha \leq_T A$. Since $\beta \leq_{wtt} \Omega \leq_{wtt} \emptyset'$, this shows that A is array computable.

Given $\alpha \leq_T A$, consider the A -computable martingale $M = \sum_p M_p$, where M_p is the martingale which has the value 2^{-p} on all strings of length up to p and then doubles the capital along the string $y = \Omega_{\alpha(p)} \upharpoonright 3p$, so that $M_p(y) = 2^p$. Note that $M(z)$ is rational for each z . If $\alpha(p) > \beta(p)$ for infinitely many p , then M Schnorr succeeds on Ω , a contradiction. \square

COROLLARY 5.2. *If A is r.e., then $\Omega \in \text{SR}^A$ iff A is r.e. traceable.*

Proof. For an r.e. set A , array computable implies r.e. traceable by the work of Ishmukhametov [5]. \square

In [6] it is shown that r.e. traceable degrees do not contain diagonally noncomputable functions, and hence, by a result of Kučera [7], the r.e. traceable degrees have measure zero. On the other hand, every real A which is Martin-Löf random relative to Ω satisfies that Ω is MR^A , by van Lambalgen's theorem [15], and hence the measure of the set of A such that Ω is SR^A is one; thus A r.e. traceable is not equivalent to $\Omega \in \text{SR}^A$. Also, $\Omega \in \text{SR}^A$ is not equivalent to A being array computable, as we now show.

The following notion of forcing appears implicitly in [4].

DEFINITION 5.3. *A tree T is a set of strings $\sigma \in 2^{<\omega}$ such that if $\sigma \in T$ and τ is a substring of σ , then $\tau \in T$. A tree T is full on a set $F \subseteq \omega$ if whenever $\sigma \in T$ and $|\sigma| \in F$, then $\sigma 0 \in T$ and $\sigma 1 \in T$. Let F_n , $n \in \omega$, be finite sets such that each F_n is an interval of ω , $|F_{n+1}| > |F_n|$, and $\bigcup_n F_n = \omega$. The sequence F_n , $n \in \omega$, is called a very strong array. Let P be the set of computable perfect trees T such that T is full on F_n for infinitely many n . Order P by $T_1 \leq_P T_2$ if $T_1 \subseteq T_2$. The partial order (P, \leq_P) is a notion of forcing that we call very strong array forcing.*

THEOREM 5.4. *For each real X there is a hyperimmune-free real A such that no real computable from X is in SR^A . In particular, as hyperimmune-free implies array computable, there is an array computable real A such that $\Omega \notin \text{SR}^A$.*

Proof. Let A be sufficiently generic for very strong array forcing. Then A is hyperimmune-free, as may be proved by modifying the standard construction of a hyperimmune-free degree [10] to work with trees that are full on infinitely many F_n , $n \in \omega$.

Moreover, for each real B computable from X , there is an n (hence infinitely many n) such that A agrees with B on F_n . Indeed, given a condition T , a condition extending T and ensuring the existence of such an n is obtained as a full subtree of T .

Hence no real B computable from X is Schnorr random relative to A . Indeed the measure of the set of those oracles B that agree with A on infinitely many F_n is zero, and it is easy to see that the measure of those B such that, for some $k > n$, A and B agree on F_k , goes to zero effectively as $n \rightarrow \infty$. Hence there is a Schnorr test relative to A which is failed by any such B , as desired. \square

QUESTION 5.5. *Characterize the (r.e.) sets of integers A such that Ω is computably random relative to A . Does this depend on the version of Ω used?*

REFERENCES

- [1] K. AMBOS-SPIES AND A. KUČERA, *Randomness in computability theory*, in *Computability Theory and Its Applications* (Boulder, CO, 1999), *Contemp. Math.* 257, AMS, Providence, RI, 2000, pp. 1–14.
- [2] B. BEDREGAL AND A. NIES, *Lowness properties of reals and hyper-immunity*, in *Proceedings of the 10th Workshop on Logic, Language, Information, and Computation*, Ouro Preto, Minas Gerais, Brazil, 2003, *Electron. Notes Theor. Comput. Sci.* 84, Elsevier, 2003, online at <http://www.elsevier.nl/locate/entcs/volume84.html>.
- [3] A. CHURCH, *On the concept of a random sequence*, *Bull. Amer. Math. Soc.*, 46 (1940), pp. 130–135.
- [4] R. G. DOWNEY, C. G. JOCKUSCH, JR., AND M. STOB, *Array nonrecursive sets and genericity*, in *Computability, Enumerability, Unsolvability: Directions in Recursion Theory*, S. B. Cooper, T. A. Slaman, and S. S. Wainer, eds., *London Math. Soc. Lecture Note Ser.*, Cambridge University Press, Cambridge, UK, 1996, pp. 93–104.
- [5] S. ISHMUKHAMETOV, *Weak recursive degrees and a problem of Spector*, *Recursion Theory and Complexity* (Kazan, 1997), *de Gruyter Ser. Log. Appl.* 2, de Gruyter, Berlin, 1999, pp. 81–87.
- [6] B. KJOS-HANSEN, W. MERKLE, AND F. STEPHAN, *Kolmogorov Complexity and the Recursion Theorem*, to appear.
- [7] A. KUČERA, *Measure, Π_1^0 -classes and complete extensions of PA*, in *Recursion Theory Week* (Oberwolfach, 1984), *Lecture Notes in Math.* 1141, Springer, Berlin, 1985, pp. 245–259.
- [8] A. KUČERA AND S. TERWIJN, *Lowness for the class of random sets*, *J. Symbolic Logic*, 64 (1999), pp. 1396–1402.
- [9] P. MARTIN-LÖF, *The definition of random sequences*, *Inform. and Control*, 9 (1966), pp. 602–619.
- [10] D. A. MARTIN AND W. MILLER, *The degrees of hyperimmune sets*, *Z. Math. Logik Grundlag. Math.*, 14 (1968), pp. 159–166.
- [11] A. NIES, *Lowness properties and randomness*, *Adv. Math.*, 197 (2005), pp. 274–305.
- [12] A. NIES, F. STEPHAN, AND S. TERWIJN, *Randomness, relativization, and Turing degrees*, *J. Symbolic Logic*, 70 (2005), pp. 515–535.
- [13] C.-P. SCHNORR, *Zufälligkeit und Wahrscheinlichkeit. Eine algorithmische Begründung der Wahrscheinlichkeitstheorie*, *Lecture Notes in Math.* 218, Springer-Verlag, Berlin, 1971.
- [14] S. TERWIJN AND D. ZAMBELLA, *Computational randomness and lowness*, *J. Symbolic Logic*, 66 (2001), pp. 1199–1205.
- [15] M. VAN LAMBALGEN, *The axiomatization of randomness*, *J. Symbolic Logic*, 55 (1990), pp. 1143–1167.
- [16] D. ZAMBELLA, *On Sequences with Simple Initial Segments*, ILLC Technical Report ML 1990-05, University of Amsterdam, Amsterdam, The Netherlands, 1990.

AN EFFICIENT ALGORITHM FOR COMPUTING OPTIMAL DISCRETE VOLTAGE SCHEDULES*

MINMING LI[†] AND FRANCES F. YAO[‡]

Abstract. We consider the problem of job scheduling on a variable voltage processor with d discrete voltage/speed levels. We give an algorithm which constructs a minimum energy schedule for n jobs in $O(dn \log n)$ time. Previous approaches solve this problem by first computing the optimal continuous solution in $O(n^3)$ time and then adjusting the speed to discrete levels. In our approach, the optimal discrete solution is characterized and computed directly from the inputs. We also show that $O(n \log n)$ time is required; hence the algorithm is optimal for fixed d .

Key words. scheduling, energy efficiency, variable voltage processor, discrete optimization

AMS subject classifications. 90B35, 90B80

DOI. 10.1137/050629434

1. Introduction. Advances in processor, memory, and communication technologies have enabled the development and widespread use of portable electronic devices. As such devices are typically powered by batteries, energy efficiency has become an important issue. With dynamic voltage scaling (DVS) techniques, processors are able to operate at a range of voltages and frequencies. Since energy consumption is at least a quadratic function of the supply voltage (hence CPU speed), it saves energy to execute jobs as slowly as possible while still satisfying all timing constraints.

We refer to the associated scheduling problem as the min-energy DVS scheduling problem (or DVS problem for short); the precise formulation will be given in section 2. The problem is different from classical scheduling on fixed-speed processors, and it has received much attention from both theoretical and engineering communities in recent years. One of the earliest theoretical models for DVS was introduced in [1]. They gave a characterization of the min-energy DVS schedule and an $O(n^3)$ algorithm¹ for computing it. No special assumption was made on the power consumption function except convexity. This optimal schedule has been referenced widely, since it provides a main benchmark for evaluating other scheduling algorithms in both theoretical and simulation work.

In the min-energy DVS schedule mentioned above, the processor must be able to run at *any* real-valued speed s in order to achieve optimality. In practice, variable voltage processors run only at a finite number of speed levels chosen from specific points on the power function curve. For example, the Intel *SpeedStep* technology [2] currently used in Intel's notebooks supports only 3 speed levels, although the new *Foxon* technology will soon enable Intel server chips to run at as many as 64 speed

*Received by the editors April 18, 2005; accepted for publication (in revised form) August 15, 2005; published electronically January 6, 2006. This work was supported in part by the Research Grants Council of Hong Kong under grant CityU122105, the National Natural Science Foundation of China under grants 60135010 and 60321002, and the Chinese National Key Foundation Research & Development Plan (2004CB318108).

<http://www.siam.org/journals/sicomp/35-3/62943.html>

[†]Department of Computer Science and Technology, State Key Laboratory of Intelligent Technology and Systems, Tsinghua University, Beijing, China (liminming98@mails.tsinghua.edu.cn).

[‡]Department of Computer Science, City University of Hong Kong, Hong Kong, China (csfyao@cityu.edu.hk).

¹The complexity of the algorithm was said to be further reducible in [1], but that claim has since been withdrawn.

grades. Thus, an accurate model for min-energy scheduling should capture the discrete, rather than continuous, nature of the available speed scale. This consideration has motivated our present work.

In this paper we consider the discrete version of the DVS scheduling problem. Denote by $s_1 > s_2 > \dots > s_d$ the clock speeds corresponding to d given discrete voltage levels. The goal is to find, under the restriction that only these speeds are available for job execution, a schedule that consumes as little energy as possible. (It is assumed that the highest speed s_1 is fast enough to guarantee a feasible schedule for the given jobs.) This problem was considered in [3] for a single job (i.e., $n = 1$), where they observed that minimum energy is achieved by using the immediate neighbors s_i, s_{i+1} of the ideal speed s in appropriate proportions. It was later extended in [4] to give an optimal discrete schedule for n jobs, obtained by first computing the optimal continuous DVS schedule and then individually adjusting the speed of each job appropriately to adjacent levels as done in [3].

The following question naturally arises: Is it possible to find a direct approach for solving the optimal discrete DVS scheduling problem without first computing the optimal continuous schedule? We answer the question in the affirmative. For n jobs with arbitrary arrival-time/deadline constraints and d given discrete supply voltages (speeds), we give an algorithm that finds an optimal discrete DVS schedule in $O(dn \log n)$ time. We also show that this complexity is optimal for any fixed d . We remark that the $O(n^3)$ algorithm for finding the continuous DVS schedule (cf. section 2) computes the highest speed, second highest speed, etc. for execution in a strictly sequential manner and may use up to n different speeds in the final schedule. Therefore it is unclear a priori how to find shortcuts to solve the discrete problem. Our approach is different from that of [4], which is based on the continuous version and therefore requires $O(n^3)$ time.

Our algorithm for optimal discrete DVS proceeds in two stages. In stage 1, the jobs in J are partitioned into d disjoint groups J_1, J_2, \dots, J_d , where J_i consists of all jobs whose execution speeds in the continuous optimal schedule S_{opt} lie between s_i and s_{i+1} . We show that this multilevel partition can be obtained without determining the exact optimal execution speed of each job. In stage 2, we proceed to construct an optimal schedule for each group J_i using two speeds s_i and s_{i+1} . Both the separation of each group J_i in stage 1, and the subsequent scheduling of J_i using two speed levels in stage 2, can be accomplished in time $O(n \log n)$ per group. Hence this two-stage algorithm yields an optimal discrete voltage schedule for J in total time $O(dn \log n)$. The algorithm admits a simple implementation, although its proof of correctness and complexity analysis are nontrivial. Aside from its theoretical value, we also expect our algorithm to be useful in generating optimal discrete DVS schedules for simulation purposes as in the continuous case.

We briefly mention some additional theoretical results on DVS, although they are not directly related to the problem considered in this paper. In [1], two on-line heuristics, average rate (AVR) and optimal available (OPA), were introduced for the case that jobs arrive one at a time. AVR was shown to have a competitive ratio of at most 8 in [1]; recently a tight competitive ratio of 4 was proven for OPA in [5]. For jobs with fixed priority, the scheduling problem was shown to be NP-hard, and an FPTAS was given in [6]. In addition, [7] gave efficient algorithms for computing the optimal schedule for job sets structured as trees. (The interested reader can find further references in these papers.)

The remainder of the paper is organized as follows. We give the problem formu-

lation and review the optimal continuous schedule in section 2. Section 3 discusses some mathematical properties associated with earliest deadline first (EDF) scheduling under different speeds. Sections 4 and 5 give details of the two stages of the algorithm as outlined above. The combined algorithm and a lower bound are presented in section 6. Finally, some concluding remarks are given in section 7.

2. Problem formulation. Each job j_k in a job set J over $[0, 1]$ is characterized by three parameters: arrival time a_k , deadline b_k , and required number of CPU cycles R_k . A schedule S for J is a pair of functions $(s(t), job(t))$ defining the processor speed and the job being executed at time t . Both functions are piecewise constant with finitely many discontinuities. A *feasible* schedule must give each job its required number of cycles between arrival time and deadline (with perhaps intermittent execution). We assume that the power P , or energy consumed per unit time, is a convex function of the processor speed. The total energy consumed by a schedule S is $E(S) = \int_0^1 P(s(t))dt$. The goal of the min-energy scheduling problem is to find, for any given job set J , a feasible schedule that minimizes $E(S)$. We refer to this problem as *DVS* scheduling (or sometimes *continuous DVS* scheduling to distinguish it from the discrete version below).

In the discrete version of the problem, we assume d discrete voltage levels are given, enabling the processor to run at d clock speeds $s_1 > s_2 > \dots > s_d$. The goal is to find a min-energy schedule for a job set using only these speeds. We may assume that, in each problem instance, the highest speed s_1 is always fast enough to guarantee a feasible schedule for the given jobs. We refer to this problem as *discrete DVS* scheduling.

For the continuous DVS scheduling problem, the optimal schedule S_{opt} can be characterized based on the notion of a *critical interval* for J , which is an interval I in which a group of jobs must be scheduled at maximum constant speed $g(I)$ in any optimal schedule for J . The algorithm proceeds by identifying such a critical interval I , scheduling those “critical” jobs at speed $g(I)$ over I and then constructing a subproblem for the remaining jobs and solving it recursively. The optimal $s(t)$ is in fact unique, whereas $job(t)$ is not always so. The details are given below.

DEFINITION 2.1. *Define the intensity of an interval $I = [z, z']$ to be*

$$g(I) = \frac{\sum R_j}{z' - z},$$

where the sum is taken over all jobs j_ℓ with $[a_\ell, b_\ell] \subseteq [z, z']$.

The interval $[c, d]$ achieving the maximum $g(I)$ will be the critical interval chosen for the current job set. All jobs $j_\ell \in J$ satisfying $[a_\ell, b_\ell] \subseteq [c, d]$ can be feasibly scheduled at speed $g([c, d])$ by the EDF principle. The interval $[c, d]$ is then removed from $[0, 1]$; all remaining intervals $[a_j, b_j]$ are updated (compressed) accordingly, and the algorithm recurses. The complete algorithm is given in Algorithm 1.

Let $CI_i \subseteq [0, 1]$ be the i th critical interval of J . Denote by Cs_i the execution speed during CI_i and by CJ_i those jobs executed in CI_i . We take note of a basic property of critical intervals which will be useful in later discussions.

LEMMA 2.2. *A job $j_\ell \in J$ belongs to $\bigcup_{k=1}^i CJ_k$ if and only if the interval $[a_\ell, b_\ell]$ of j_ℓ satisfies $[a_\ell, b_\ell] \subseteq \bigcup_{k=1}^i CI_k$.*

Proof. The “if” direction is straightforward. For the “only if” part, it can be proven by induction on i that $j_\ell \in CJ_i$ implies $[a_\ell, b_\ell] \subseteq \bigcup_{k=1}^i CI_k$, based on the way critical intervals are successively chosen. \square

Algorithm 1 *OS (Optimal Schedule)***Input:** a job set J **Output:** Optimal Voltage Schedule S **repeat** Select $I^* = [z, z']$ with $s = \max g(I)$ Schedule $j_i \in J_{I^*}$ at s over I^* by Earliest Deadline First policy $J \leftarrow J - J_{I^*}$ **for all** $j_k \in J$ **do** **if** $b_k \in [z, z']$ **then** $b_k \leftarrow z$ **else if** $b_k \geq z'$ **then** $b_k \leftarrow b_k - (z' - z)$ **end if**

Reset arrival times similarly

end for**until** J is empty

3. EDF with variable speeds. The EDF principle defines an ordering on the jobs according to their deadlines. At any time t , among jobs j_k that are available for execution, that is, j_k satisfying $t \in [a_k, b_k)$ and j_k not yet finished by t , it is the job with minimum b_k that will be executed during $[t, t + \epsilon]$. EDF is a natural scheduling principle, and many optimal schedules (such as the continuous min-energy schedule described above) in fact conform to it. All schedules considered in the remainder of this paper are EDF schedules. Hence we assume the jobs in $J = \{j_1, \dots, j_n\}$ are indexed by their deadlines.

We introduce an important tool for solving the discrete DVS scheduling problem: an EDF schedule that runs at some constant speed s (except for periods of idleness).

DEFINITION 3.1. *An s -schedule for J is a schedule which conforms to the EDF principle and uses constant speed s in executing any job of J .*

As long as there are unfinished jobs available at time t , an s -schedule will select a job by the EDF principle and execute it at speed s . An s -schedule may contain periods of idleness when there are no jobs available for execution. An s -schedule may also yield an unfeasible schedule for J since the speed constraint may leave some jobs unfinished by their deadlines.

DEFINITION 3.2. *In any schedule S , a maximal subinterval of $[0, 1]$ devoted to executing the same job j_k is called an execution interval (for j_k with respect to S). Denote by $I_k(S)$ the collection of all execution intervals for j_k with respect to S . With respect to the s -schedule for J , any execution interval will be called an s -execution interval, and the collection of all s -execution intervals for job j_k will be denoted by I_k^s .*

Notice that for any EDF schedule S , it is always true that $I_i(S) \subseteq [a_i, b_i] - \cup_{k=1}^{i-1} I_k(S)$. For a given J , we observe some interesting monotone relations that exist among the EDF schedules of J with respect to different speed functions. These relations will be exploited by our algorithms later. They may also be of independent interest in studying other types of scheduling problems.

LEMMA 3.3. *Let S_1 and S_2 be two EDF schedules whose speed functions satisfy $s_1(t) > s_2(t)$ for all t whenever S_1 is not idle.*

(1) *For any t and any job j_k , the workload of j_k executed by time t under S_1 is*

always no less than that under S_2 .

(2) $\cup_{k=1}^i I_k(S_1) \subseteq \cup_{k=1}^i I_k(S_2)$ for any i , $1 \leq i \leq n$.

(3) Any job of J that can be finished under S_2 is always finished strictly earlier under S_1 .

(4) If S_2 is a feasible schedule for J , then so is S_1 .

Proof. We prove (1) and (2) by induction on i . When $i = 1$, since both S_1 and S_2 start executing j_1 at time a_1 , it is easy to see that induction hypotheses (1) and (2) are both true. Assume that they hold for jobs j_1, \dots, j_{i-1} ; we will prove (1) and (2) for j_i . The time V_1 available for executing j_i under S_1 is $V_1 = [a_i, b_i] - \cup_{k=1}^{i-1} I_k(S_1)$, which satisfies $V_1 \supseteq V_2$ for the corresponding available time under S_2 because of induction hypothesis (2). This together with the assumption $s_1(t) > s_2(t)$ proves (1); that is, the execution of j_i by S_1 will always be ahead of that by S_2 . Assuming that S_2 finishes j_i at time t , then we have $I_i(S_1) \subseteq [a_i, t] \subseteq \cup_{k=1}^i I_k(S_2)$ from which (2) follows inductively. \square

Note that as a special case, Lemma 3.3 holds when we substitute s_1 -schedule and s_2 -schedule, with $s_1 > s_2$, for S_1 and S_2 , respectively.

LEMMA 3.4. *The s -schedule for J contains at most $2n$ s -execution intervals and can be computed in $O(n \log n)$ time if the arrival times and deadlines are already sorted.*

Proof. The end of an execution interval corresponds to the moment when either a job is finished or a new job arrives. There can be at most $2n$ such endpoints, and hence at most $2n$ s -execution intervals. If the arrival times and deadlines are already sorted, then generating one s -execution interval costs $O(\log n)$ time, and the entire schedule can be computed in $O(n \log n)$ time. This completes the proof. \square

4. Partition of jobs by speed level. We will consider the first stage of the algorithm for optimal discrete DVS in this section. Clearly, to get an $O(dn \log n)$ -time partition of J into d groups corresponding to d speed levels, it suffices to give an $O(n \log n)$ algorithm which can properly separate J into two groups according to any given speed s .

DEFINITION 4.1. *Given a job set J and any speed s , let $J^{\geq s}$ and $J^{< s}$ denote the subset of J consisting of jobs whose executing speeds are $\geq s$ and $< s$, respectively, in the (continuous) optimal schedule of J . We refer to the partition $\langle J^{\geq s}, J^{< s} \rangle$ as the s -partition of J .*

Let $T^{\geq s} \subseteq [0, 1]$ be the union of all critical intervals CI_i with $Cs_i \geq s$. By Lemma 2.2, a job i is in $J^{\geq s}$ if and only if its interval $[a_i, b_i] \subseteq T^{\geq s}$. Thus $J^{\geq s}$ is uniquely determined by $T^{\geq s}$, and we can focus on computing $T^{\geq s}$ instead. Let $T^{< s} = [0, 1] - T^{\geq s}$, and we refer to $\langle T^{\geq s}, T^{< s} \rangle$ as the s -partition of time for J .

An example of J with 11 jobs is given in Figure 1, together with the optimal speed function $S_{opt}(t)$. The portion of $S_{opt}(t)$ lying above the horizontal line $Y = s$ projects to $T^{\geq s}$ on the time axis. In general, $T^{\geq s}$ may consist of a number of connected components.

In the remainder of this section, we will show that certain features existing in the s -schedule of J can be used for identifying connected components of $T^{< s}$. This then leads to an efficient algorithm for computing the s -partition of time $\langle T^{\geq s}, T^{< s} \rangle$.

DEFINITION 4.2. *In the s -schedule for J , we say a deadline b_i is tight if job j_i is either unfinished at time b_i or is finished just on time at b_i . An idle interval $g = [t, t']$ in the s -schedule is called a gap.*

Figure 2 depicts the s -schedule for the sample job set J considered in Figure 1. All tight deadlines and gaps have been marked along the time axis. By overlaying the

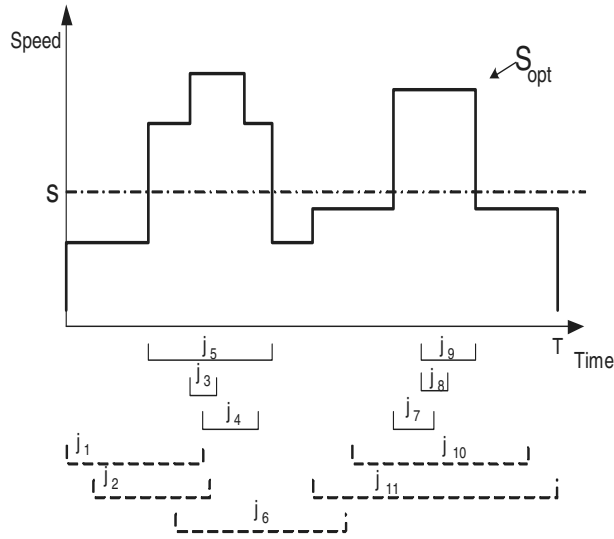


FIG. 1. The s -partition for a sample J . The jobs are represented by their intervals only and indexed according to deadline. Solid intervals represent jobs belonging to $J^{\geq s}$, while dashed intervals represent jobs belonging to $J^{< s}$.

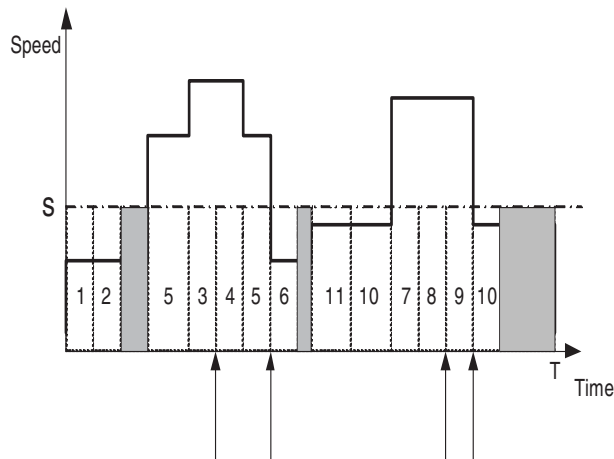


FIG. 2. The s -execution intervals for the same J in Figure 1 are illustrated, where the number indicates which job is being executed. Shaded blocks correspond to gaps (idle time), while arrows point to tight deadlines.

s -partition of time $\langle T^{\geq s}, T^{< s} \rangle$ for J , we notice that (1) tight deadlines exist only in $T^{\geq s}$, and (2) each connected component of $T^{\geq s}$ ends with a tight deadline. We prove below that these properties always hold for any job set.

LEMMA 4.3. (1) *Tight deadlines in an s -schedule can exist only in $T^{\geq s}$.*

(2) *The rightmost point of each connected component of $T^{\geq s}$ must be a tight deadline.*

Proof. As observed before, the only jobs available for execution during $T^{< s}$ are those from $J^{< s}$. Furthermore, by property (3) of Lemma 3.3, all jobs of $J^{< s}$ will be finished strictly before their deadlines under the s -schedule, thus yielding no tight

deadlines in $T^{<s}$. This proves property (1). For property (2), we argue that it is a consequence of property (1) in Lemma 3.3. Let b_i be the end of a connected component of $T^{\geq s}$. Job j_i , being the last executed job of a critical interval, is finished exactly on time under S_{opt} (which runs at speed at least s throughout $T^{\geq s}$). Therefore, job j_i must have a tight deadline under the s -schedule. \square

Property (2) of Lemma 4.3 gives a necessary condition for identifying the right boundary of each connected component of $T^{\geq s}$. The corresponding left boundary of such a component can also be identified through left-right symmetry of the scheduling problem with respect to time.

DEFINITION 4.4. *Given a job set J , the reverse job set J^{rev} consists of jobs with the same workload but time intervals $[1 - b_i, 1 - a_i]$. The s -schedule for J^{rev} is called the reverse s -schedule for J . We call an arrival time a_i (for the original job set J) tight if $1 - a_i$ corresponds to a tight deadline in the reverse s -schedule for J .*

One may also view the reverse s -schedule as a schedule which runs backwards: starting from time 1 and executing jobs of J by the latest arrival time first principle at constant speed s whenever possible. Lemma 4.5 is the symmetric analogue of Lemma 4.3.

LEMMA 4.5. (1) *Tight arrival times in an s -schedule can exist only in $T^{\geq s}$.*

(2) *The leftmost point of each connected component of $T^{\geq s}$ must be a tight arrival time.*

Lemmas 4.3 and 4.5 are not sufficient by themselves to enable an efficient separation of $T^{\geq s}$ from $T^{<s}$. Fortunately, we have an additional useful property related to $T^{<s}$. Observe that in Figure 2 all gaps of the s -schedule fall within $T^{<s}$. This is in fact true in general, and, furthermore, a gap must exist in $T^{<s}$ as we prove next.

LEMMA 4.6. *Gaps in an s -schedule can exist only in $T^{<s}$; furthermore, a gap must exist in $T^{<s}$.*

Proof. Suppose a gap in an s -schedule occurs at some time $t \in T^{\geq s}$; that is, all jobs $J(t)$ in J whose intervals overlap t have been finished. In particular, no jobs belonging to $J(t) \cap J^{\geq s}$ are available. Since the schedule s_{opt} runs at higher speed than s over $T^{\geq s}$ in executing $J^{\geq s}$, it must also finish all jobs of $J(t) \cap J^{\geq s}$ before time t by Lemma 3.3. In other words, s_{opt} would have a gap at time t which is not possible. This proves that gaps can exist only in $T^{<s}$. For the second part, we note that the total workload of $J^{<s}$, which is executed over $T^{<s}$, is less than $s \cdot |T^{<s}|$; hence a gap must exist. \square

Finally, we collect the properties that will be used by the partition algorithm in the following theorem. We first give a definition.

DEFINITION 4.7. *Given a gap $[x, y]$ in an s -schedule, we define the expansion of $[x, y]$ to be the smallest interval $[b, a]$ satisfying (1) $[b, a] \supseteq [x, y]$, and (2) b and a are tight deadline and tight arrival time, respectively, of the s -schedule. (Note: we adopt the convention that 0 is considered a tight deadline, while 1 is considered a tight arrival time.) See Figure 3.*

THEOREM 4.8. (1) *A gap always exists in an s -schedule if $T^{<s} \neq \emptyset$.*

(2) *The expansion $[b, a]$ of a gap $[x, y]$ defines the connected component in $T^{<s}$ containing $[x, y]$.*

Proof. Property (1) comes from Lemma 4.6, while property (2) follows from Lemmas 4.3 and 4.5. \square

Notice that although Theorem 4.8 guarantees that one can always find a gap and then use it to identify a connected component C of $T^{<s}$ (provided $T^{<s} \neq \emptyset$), it is not true that *all* connected components of $T^{<s}$ must contain gaps and can be identified

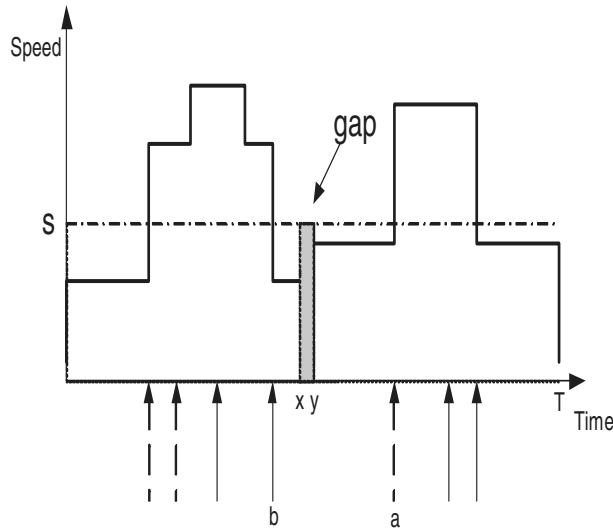


FIG. 3. Gap expansion: the indicated gap will be expanded into $[b, a]$, a connected component of $T^{<s}$.

simultaneously. However, once a component C is found, by deleting the s -execution intervals of all jobs whose interval $[a_i, b_i]$ intersects with C , gaps can surely be found (provided $T^{<s} - C \neq \emptyset$), and the process can continue. This is true because, by reasoning similar to that of Lemma 4.6, the total workload of the remaining jobs in $J^{<s}$ over $T^{<s} - C$ is less than $s \cdot |T^{<s} - C|$; hence a gap must exist.

The detailed algorithm for generating the s -partition is given in Algorithm 2 below.

THEOREM 4.9. *Algorithm 2 finds the s -partition $\langle J^{\geq s}, J^{<s} \rangle$ for a job set J in $O(n \log n)$ time.*

Proof. The correctness of the algorithm is based on Theorem 4.8 and the discussions following the theorem. For the complexity part, sorting and generating s -schedules take $O(n \log n)$ time. We now analyze individual steps inside the for loop. For step 1, finding the expansion of a gap takes only $O(\log n)$ time by binary search; with at most n expansions (to find at most n connected components) the total cost is $O(n \log n)$. Step 2 can be done, with standard data structures such as interval trees, in time $O(\log n) + |J_{new}^{<s}|$, which amounts to total time $O(n \log n)$ since $\sum |J_{new}^{<s}| = O(n)$. It remains to consider steps 7 and 8. Since each individual gap is added to and deleted from the sorted list $Gaps$ only once, and since there are at most $2n$ s -execution intervals (hence gaps), the cost is at most $O(n \log n)$. This shows that the total running time of Algorithm 2 is $O(n \log n)$. \square

We next use Algorithm 2 as a subroutine to obtain Algorithm 3.

THEOREM 4.10. *Algorithm 3 partitions job set J into d subsets corresponding to d speed levels in time $O(dn \log n)$.*

5. Two-level schedule. After Algorithm 3 completes the multilevel partition of J into subsets J_1, \dots, J_d , we can proceed to schedule the jobs in each subset J_i with two appropriate speed levels s_i and s_{i+1} . We will present a two-level scheduling algorithm whose complexity is $O(n \log n)$ for a set of n jobs. For this purpose, it suffices to describe how to schedule the subset J_1 with two available speeds s_1 and

Algorithm 2 Bipartition**Input:** job set J and speed s **Output:** s -partition of J

Sort arrival times and deadlines

Generate the s -schedule and reverse s -schedule for J $J^{\geq s} \leftarrow J$ $J^{< s} \leftarrow \emptyset$ $T^{\geq s} \leftarrow [0, 1]$ $T^{< s} \leftarrow \emptyset$ $Gaps =$ sorted list of gaps in s -schedule**while** $Gaps \neq \emptyset$ **do**

1. Choose any gap $[x, y]$ from $Gaps$. Find the expansion $[b, a]$ of $[x, y]$.
2. $J_{new}^{< s} = \{ \text{all jobs in } J^{\geq s} \text{ whose interval } [a_j, b_j] \text{ intersects with } [b, a] \}$
3. $J^{\geq s} \leftarrow J^{\geq s} - J_{new}^{< s}$
4. $J^{< s} \leftarrow J^{< s} \cup J_{new}^{< s}$
5. $T^{\geq s} \leftarrow T^{\geq s} - [b, a]$
6. $T^{< s} \leftarrow T^{< s} \cup [b, a]$
7. $Gaps = Gaps \cup \{ s\text{-execution intervals of jobs in } J_{new}^{< s} \}$
8. Delete all gaps that are contained in $[b, a]$

end whileReturn $J^{< s}$ and $J^{\geq s}$ **Algorithm 3** Multilevel Partition**Input:**job set J and speed $s_1 > \dots > s_d > s_{d+1} = 0$ **Output:**Partition of J into J_1, \dots, J_d corresponding to speed levels**for** $i = 1$ to d **do**Obtain $J^{\geq s_{i+1}}$ from J using Algorithm 2 $J_i \leftarrow J^{\geq s_{i+1}}$ $J \leftarrow J - J_i$ Update J as in Algorithm 1**end for**

s_2 , where $s_1 > s_2 > 0$. We will schedule each connected component of J_1 separately. Thus, the two-level scheduler deals only with “eligible” input job sets, i.e., those with a continuous optimal schedule speed $s_{opt}(t)$ satisfying $s_1 \geq s_{opt}(t) \geq s_2$ for all t . (Clearly, this condition is satisfied by each connected component of $J_1 = J^{\geq s_2}$ output from Algorithm 3.) We give an alternative and equivalent definition of “eligibility” in the following. This definition does not make reference to $s_{opt}(t)$, and hence is more useful for the purpose of deriving a two-level schedule directly.

DEFINITION 5.1. For a job set J over $[0, 1]$, a two-level schedule with speeds s_1 and s_2 (or (s_1, s_2) -schedule for short) for J is a feasible schedule $s(t)$ for J , which is piecewise constant over $[0, 1]$ with either $s(t) = s_1$ or $s(t) = s_2$ for any t .

In other words, an (s_1, s_2) -schedule for J is a schedule using only speeds s_1 and s_2 which finishes every job and leaves no idle time.

LEMMA 5.2. For a job set J over $[0, 1]$, an (s_1, s_2) -schedule exists if and only if (1) the s_1 -schedule for J is a feasible schedule, and

(2) the s_2 -schedule for J contains no idle time in $[0, 1]$.

Proof. The “only if” direction is easy to see. Suppose a two-level schedule $s(t)$ exists with $s_1 \geq s(t) \geq s_2$ for all $t \in [0, 1]$. It follows from Lemma 3.3 that with speed s_1 the processor can finish all jobs just as with speed $s(t)$, while with speed s_2 the processor will not finish any job earlier than with speed $s(t)$, and hence will never be idle. This proves (1) and (2). For the “if” direction, suppose (1) and (2) both hold. Because the s_1 -schedule generates a feasible schedule, the optimal continuous schedule $s_{opt}(t)$ must satisfy $s_{opt}(t) \leq s_1$ for all $t \in [0, 1]$. On the other hand, (2) implies $J = J^{\geq s_2}$ by Lemma 4.6; that is, $s_{opt}(t) \geq s_2$ for all $t \in [0, 1]$. Using the result in [4], we can first calculate the continuous optimal schedule $s_{opt}(t)$ for J and then adjust the execution speed s of each job to be a combination of s_1 or s_2 in the right proportion to achieve the same average speed s . This results in a two-level schedule for J with speeds s_1 and s_2 . \square

In view of the preceding lemma, we give the following definition of eligibility for input job sets to two-level scheduling.

DEFINITION 5.3. A job set J over $[0, 1]$ is said to be eligible for (s_1, s_2) -scheduling if

- (1) the s_1 -schedule for J is a feasible schedule, and
- (2) the s_2 -schedule for J contains no idle time in $[0, 1]$.

We will consider only eligible job sets in discussing two-level scheduling in the remainder of this section. An (s_1, s_2) -schedule for J is said to be *optimal* if it consumes minimum energy among all (s_1, s_2) -schedules for J .

LEMMA 5.4. All (s_1, s_2) -schedules for an eligible job set J consume the same amount of energy, and hence are optimal.

Proof. The energy consumption is determined by the total amount of time the processor runs at speeds s_1 and s_2 , respectively. Suppose, in an optimal schedule for J , that α time is devoted to speed s_1 and β time is devoted to speed s_2 . An optimal schedule will not contain any idle period; hence the following equations are satisfied:

$$\begin{cases} \alpha s_1 + \beta s_2 = \sum R_i, \\ \alpha + \beta = 1. \end{cases}$$

Clearly, any (s_1, s_2) -schedule for J will also satisfy the above two equations. Since these equations uniquely determine α and β , the lemma follows. \square

The two-level schedule as described in the proof of Lemma 5.2, which first computes the continuous optimal schedule and then rounds the execution speed of each job up and down appropriately [4], requires $O(n^3)$ computation time. We now describe a more efficient algorithm which directly outputs a two-level schedule without first computing the continuous optimal schedule. The algorithm runs in $O(n)$ time if the input jobs are already sorted by deadline (as obtained via Algorithm 3) and $O(n \log n)$ time in general.

The two-level scheduling algorithm (Algorithm 4) proceeds as follows. It first computes the s_2 -schedule for J which in general does not provide a feasible schedule. We then transform it into a feasible schedule by suitably adjusting the execution speed of each job from s_2 to s_1 and possibly extending its execution interval if necessary. These adjustments are done in an orderly and systematic manner to ensure overall feasibility. The algorithm needs to consult the corresponding s_1 -schedule of J in making the transformation. An (s_1, s_2) -schedule for J is produced at the end which by Lemma 5.4 is an optimal two-level schedule.

Algorithm 4 Two-Level Schedule**Input:**

speeds s_1, s_2 , where $s_1 > s_2$

An eligible job set J for (s_1, s_2) -scheduling

Variables:

Committed: the list of allocated time intervals.

Committed(i): the time intervals allocated to job j_i .

Output:

Optimal (s_1, s_2) -schedule for J

Compute s_1 -schedule for J to obtain $I_k^{s_1}$ for $k = 1, \dots, n$.

Compute s_2 -schedule for J to obtain $I_k^{s_2}$ for $k = 1, \dots, n$.

$Committed \leftarrow \emptyset$

for $i = n$ **downto** 1 **do**

1. $I = I_i^{s_2} - Committed$

2. Take $I' \subseteq I_i^{s_1}$ of appropriate length (possibly 0) from the right end of $I_i^{s_1}$ to obtain an (s_1, s_2) -schedule for j_i over $I \cup I'$

3. $Committed(i) = I \cup I'$

4. $Committed \leftarrow Committed \cup Committed(i)$

end for

5.1. Correctness of two-level scheduling algorithm. Let J be an eligible job set for (s_1, s_2) -scheduling. We will show that Algorithm 4 indeed outputs an (s_1, s_2) -schedule for J .

The jobs in J are sorted in increasing order by their deadlines as j_1, j_2, \dots, j_n . After computing the s_1 -schedule and s_2 -schedule for J , the algorithm then allocates appropriate execution time and speed for each job j_i in the order $i = n, \dots, 1$. Step 2 of the for loop carries out the allocation for job j_i . We examine this step in more detail in the following lemma.

LEMMA 5.5. *In step 2 of the for loop, by choosing an appropriate interval $I' \subseteq I_i^{s_1}$ (assuming $I_i^{s_1} \cap Committed = \emptyset$), an (s_1, s_2) -schedule for job j_i over $I \cup I'$ can be found where $I = I_i^{s_2} - Committed$.*

Proof. There are two cases to consider when step 2 is encountered. Suppose job j_i can be feasibly scheduled with speed s_1 over $I = I_i^{s_2} - Committed$. Since the s_2 -schedule of j_i over I clearly has no idle time, Lemma 5.2 ensures that an (s_1, s_2) -schedule exists for job j_i over I . Suppose j_i cannot be feasibly scheduled at s_1 over I . Under the assumption $I_i^{s_1} \cap Committed = \emptyset$, we can take sufficient length of time I' from $I_i^{s_1}$ so that j_i can be finished at s_1 over $I \cup I'$. Therefore one can always find an (s_1, s_2) -schedule for job j_i over $I \cup I'$. \square

We next prove that the assumption $I_i^{s_1} \cap Committed = \emptyset$ in Lemma 5.5 is indeed satisfied when step 2 is encountered in the i th iteration (see property (3) below). In fact, we show by induction on i that the following induction hypotheses are maintained by the algorithm at the start of the i th iteration for $i = n, \dots, 1$.

LEMMA 5.6. *At the beginning of the i th iteration of the for loop, the following are true:*

- (1) $Committed(i+1) \subseteq I_{i+1}^{s_1} \cup I_{i+1}^{s_2}$.
- (2) $\cup_{k=i+1}^n I_k^{s_2} \subseteq Committed \subseteq (\cup_{k=i+1}^n I_k^{s_1}) \cup (\cup_{k=i+1}^n I_k^{s_2})$.
- (3) $Committed \cap (\cup_{k=1}^i I_k^{s_1}) = \emptyset$.

Proof. It is easy to verify that all three induction hypotheses hold initially for $i = n$. Now assume that they hold for iterations $i+1, \dots, n$; we will prove them for the i th iteration. Property (1) is a result of how $Committed(i+1)$ is selected as discussed in Lemma 5.5. For property (2), the right side follows from (1) since $Committed = \cup_{k=i+1}^n Committed(k)$. The left side follows from the fact that, by Lemma 5.5, each $Committed(k)$ always uses up all remaining time in $I_k^{s_2}$ not already committed to previous jobs. To prove (3), let $V = \cup_{k=1}^i I_k^{s_1}$. First, note that V is disjoint from $\cup_{k=i+1}^n I_k^{s_1}$. Next, V is contained in $\cup_{k=1}^i I_k^{s_2}$ by property (2) of Lemma 3.3; hence V is disjoint from $\cup_{k=i+1}^n I_k^{s_2}$. Thus it follows from (2) that $Committed \cap V = \emptyset$. \square

THEOREM 5.7. *Given an eligible job set J for (s_1, s_2) -scheduling, Algorithm 4 generates an (s_1, s_2) -schedule for J .*

Proof. Each job j_i is feasibly executed, with no idle time, over $Committed(i)$ at speeds $\{s_1, s_2\}$ as specified in Lemma 5.5. By the time the algorithm terminates, $Committed = \cup_{k=1}^n Committed(k) \supseteq \cup_{k=1}^n I_k^{s_2} = [0, 1]$ by property (2) of Lemma 5.6. Hence there is no idle time in $[0, 1]$. The resulting schedule thus satisfies the requirements of an (s_1, s_2) -schedule for J . \square

5.2. Complexity of two-level scheduling algorithm. We will show that the cost of Algorithm 4 is $O(n \log n)$. The algorithm first computes the s_1 -schedule and s_2 -schedule for J in $O(n \log n)$ time. The resulting list L_{s_1} with at most $2n$ s_1 -execution intervals is already sorted, and similarly for the list L_{s_2} of s_2 -execution intervals. A sorted list L_{s_1, s_2} , of size at most $4n$, representing $L_{s_1} \cap L_{s_2}$ can be obtained with cost $O(n)$. Using appropriate pointers from lists L_{s_1} , L_{s_2} , and $Committed$ into L_{s_1, s_2} , each step of the for loop can be carried out in constant time plus the number of intervals visited. Execution of step 2 may cause a splitting of some interval in L_{s_1} (to represent interval I') and corresponding splitting in L_{s_1} , L_{s_1, s_2} , and $Committed$. As only a single split can be introduced in each iteration, the overall effect is only $O(n)$. Also each subinterval in L_{s_1} , L_{s_2} , and L_{s_1, s_2} needs to be visited only once. Hence the total running time of the algorithm is $O(n)$ if the input jobs are sorted (as output by Algorithm 3). We have proved the following theorem.

THEOREM 5.8. *Algorithm 4 computes an optimal two-level schedule for J in $O(n \log n)$ time.*

Algorithm 5 Optimal Discrete DVS Schedule

Input:

job set J

speed levels: $s_1 > s_2 > \dots > s_d > s_{d+1} = 0$

Output:

Optimal Discrete DVS Schedule for J

Generate J_1, J_2, \dots, J_d by Algorithm 3

for $i = 1$ to d **do**

Schedule jobs in J_i using Algorithm 4 with speeds s_i and s_{i+1}

end for

The union of the schedules give the optimal Discrete DVS schedule for J

6. Optimal discrete voltage schedule.

THEOREM 6.1. *Algorithm 5 generates a min-energy discrete DVS (MDDVS) schedule with d voltage levels in time $O(dn \log n)$ for n jobs.*

Proof. This is a direct consequence of Theorems 4.10, 5.7, and 5.8. \square

We next show that the running time of Algorithm 5 is optimal by proving an $\Omega(n \log n)$ lower bound in the algebraic decision tree model.

THEOREM 6.2. *Any deterministic algorithm for computing an MDDVS schedule with $d \geq 2$ voltage levels will require $\Omega(n \log n)$ time for n jobs.*

Proof. The integer element uniqueness (IEU) problem is known to have $\Omega(n \log n)$ computational complexity in the algebraic decision tree model [8]. We now make a linear reduction from IEU to MDDVS. Suppose the given instance of IEU consists of n positive integers $\{x_1, x_2, \dots, x_n\}$. First, compute $N = \max\{x_i\}$ in linear time. We construct a job set $J = \{j_1, j_2, \dots, j_n\}$ over time span $[0, N]$ with $[a_i, b_i] = [x_i - 1, x_i]$ and $R_i = 1$. (The time span can be normalized to $[0, 1]$ by scaling all numbers appropriately.) Thus the time intervals of all the jobs are disjoint if and only if the integers x_i are distinct. Set the available speed levels to be $s_1 = n$ (to guarantee feasibility) and $s_d = 1$, while s_2, \dots, s_{d-1} may be any values in between. It is easy to see that the answer to the IEU problem is yes (all integers x_i are distinct) if and only if $MDDVS \leq n$ (by executing every job at speed $s_d = 1$). This completes the reduction. \square

7. Conclusion. In this paper we considered the problem of job scheduling on a variable voltage processor with d discrete voltage/speed levels. We give an algorithm which constructs a minimum energy schedule for n jobs in $O(dn \log n)$ time, which is optimal for fixed d . The min-energy discrete schedule is obtained without first computing the continuous optimal solution. Our algorithm consists of two stages: a multilevel partition of J into d disjoint groups J_i , followed by finding a two-level schedule for each J_i using speeds s_i and s_{i+1} . The individual modules in our algorithm, such as the multilevel partition and two-level scheduling, may be of interest in and of themselves aside from the main result. Our algorithm admits a simple implementation, although its proof of correctness and complexity analysis are nontrivial. We have also discovered some nice fundamental properties associated with EDF scheduling under variable speeds. Some of these properties are stated as lemmas in section 3 for easy reference. Our results may provide some new insights and tools for the problem of min-energy job scheduling on variable voltage processors. Aside from the theoretical value, we also expect the algorithm to be useful in generating optimal discrete schedules for simulation purposes as in the continuous case.

REFERENCES

- [1] F. YAO, A. DEMERS, AND S. SHENKER, *A scheduling model for reduced CPU energy*, in Proceedings of the 36th Annual IEEE Symposium on Foundations of Computer Science, 1995, pp. 374–382.
- [2] INTEL CORPORATION, *Wireless Intel SpeedStep Power Manager - Optimizing power consumption for the intel PXA27x processor family*, Wireless Intel SpeedStep(R) Power Manager White paper, 2004.
- [3] T. ISHIHARA AND H. YASUURA, *Voltage scheduling problem for dynamically variable voltage processors*, in Proceedings of the International Symposium on Low Power Electronics and Design, ACM, New York, 1998, pp. 197–201.
- [4] W. KWON AND T. KIM, *Optimal voltage allocation techniques for dynamically variable voltage processors*, ACM Transactions on Embedded Computing Systems, 4 (2005), pp. 211–230.

- [5] N. BANSAL, T. KIMBREL, AND K. PRUHS, *Dynamic speed scaling to manage energy and temperature*, in Proceedings of the 45th Annual IEEE Symposium on Foundations of Computer Science, 2004, pp. 520–529.
- [6] H. S. YUN AND J. KIM, *On energy-optimal voltage scheduling for fixed-priority hard real-time systems*, ACM Transactions on Embedded Computing Systems, 2 (2003), pp. 393–430.
- [7] M. LI, J. B. LIU, AND F. F. YAO, *Min-energy voltage allocation for tree-structured tasks*, in Proceedings of the Eleventh International Computing and Combinatorics Conference, Springer-Verlag, Berlin, 2005, pp. 283–296.
- [8] A. C.-C. YAO, *Lower bounds for algebraic computation trees with integer inputs*, SIAM J. Comput., 20 (1991), pp. 655–668.

A COMBINATORIAL LOGARITHMIC APPROXIMATION ALGORITHM FOR THE DIRECTED TELEPHONE BROADCAST PROBLEM*

MICHAEL ELKIN[†] AND GUY KORTSARZ[‡]

Abstract. Consider a synchronous network of processors, modeled by directed or undirected graph $G = (V, E)$, in which in each round every processor is allowed to choose one of its neighbors and to send a message to this neighbor. Given a processor $s \in V$ and a subset $T \subseteq V$ of processors, the *telephone multicast* problem requires computing the shortest schedule (in terms of the number of rounds) that delivers a message from s to all the processors of T . The particular case $T = V$ is called the *telephone broadcast* problem.

These problems have multiple applications in distributed computing. Several approximation algorithms with polylogarithmic ratio, including one with logarithmic ratio, for the *undirected* variants of these problems are known. However, all these algorithms involve solving large linear programs. Devising a polylogarithmic approximation algorithm for the directed variants of these problems is an open problem, posed by Ravi in [*Proceedings of the 35th Annual IEEE Symposium on Foundations of Computer Science* (FOCS '94), 1994, pp. 202–213].

We devise a *combinatorial logarithmic* approximation algorithm for these problems that applies also for the *directed broadcast* problem. Our algorithm has significantly smaller running time and seems to reveal more information about the combinatorial structure of the solution than the previous algorithms that are based on linear programming.

We also improve the lower bounds on the approximation threshold of these problems. Both problems are known to be $3/2$ -inapproximable. For the undirected (resp., directed) broadcast problem we show that it is NP-hard (resp., impossible unless $NP \subseteq DTIME(n^{O(\log n)})$) to approximate it within a ratio of $3 - \epsilon$ for any $\epsilon > 0$ (resp., $\Omega(\sqrt{\log n})$).

Key words. directed, multicast, approximation, graph

AMS subject classifications. 68W25

DOI. 10.1137/S0097539704440740

1. Introduction. Consider a network of processors modeled by a directed or undirected n -vertex graph $G = (V, E)$. Assume that the communication in the network is synchronous, i.e., occurs in discrete “rounds,” and in every round every processor is allowed to pick one of its neighbors and to send a message to this neighbor. The *telephone broadcast* problem requires computing a schedule with a minimal number of rounds that delivers a message from a given single processor, which generates the message, to all the remaining processors in the network. A more general *telephone k -multicast* problem accepts as input also a set of terminals $T \subseteq V$ of size $|T| = k$ and requires computing the shortest schedule that delivers the message to all the processors of T , whereas the processors of $V \setminus T$ may be left uninformed.

The telephone broadcast and multicast are basic primitives in distributed computing and computer communication theory, and they are used as building blocks for various more complicated tasks in these areas (see, e.g., [HHL88]). The optimization

*Received by the editors February 6, 2004; accepted for publication (in revised form) July 11, 2005; published electronically January 27, 2006. A preliminary version of this paper appeared in *Proceedings of the 34th Annual ACM Symposium on Theory of Computing*, 2002, pp. 438–447.

<http://www.siam.org/journals/sicomp/35-3/44074.html>

[†]Department of Computer Science, Yale University, New Haven, CT 06520-8285 (elkin@cs.yale.edu). This author’s work was supported by the Department of Defense University Research Initiative (URI), administered by the Office of Naval Research under grant N00014-01-1-0795. Part of this author’s work was done at the School of Mathematics, Institute for Advanced Study, Princeton, NJ.

[‡]Department of Computer Science, Rutgers University, Camden, NJ (guyk@crab.rutgers.edu).

variants of the broadcast and multicast primitives were intensively studied during the last decade [BGN+98, KP95, R94, S00, F01].

Kortsarz and Peleg [KP95] devised the first approximation algorithm for the *undirected broadcast* problem. Their algorithm constructs schedules that can be longer than the optimal schedule by $\Theta(\sqrt{n})$ for the instance at hand. In a breakthrough paper [R94], Ravi devised an algorithm that provides an $O(\frac{\log^2 k}{\log \log k})$ -approximation for the *undirected multicast* problem. This result obviously implies an $O(\frac{\log^2 n}{\log \log n})$ -approximation for the undirected broadcast problem. Bar-Noy et al. [BGN+98] improved this result and devised an algorithm that provides logarithmic approximation guarantees for the *undirected* multicast and broadcast problems.

Note that all these algorithms do not apply for the *directed* versions of these problems. The importance of designing algorithms for telephone broadcast on directed graphs was realized already some 10 years ago by Ravi, who placed this problem in the first place in the list of open problems in the Conclusion section of [R94].

Also, both known approximation algorithms that provide polylogarithmic guarantees for these problems [R94, BGN+98] use large linear programming (LP). To the best of our knowledge, the worst-case running time for solving the linear program of [BGN+98] is no smaller than $\Omega(n^6)$. The algorithm of [R94] is significantly faster (even though it is also an LP-based algorithm) and requires $\tilde{O}(|E||V|^2)$ expected time¹ or $\tilde{O}(|E||V|^3)$ deterministic time. However, its approximation guarantee is superlogarithmic.

In this paper we devise a deterministic *combinatorial* $O(\log n)$ ratio algorithm for the telephone broadcast problem that applies to the *directed broadcast* problem as well. For the undirected case, the algorithm has $O(\log k)$ ratio for the k -multicast problem. The worst-case running time of our algorithm is $\tilde{O}(|E||V|)$, which is a significant improvement over the worst-case running times of [BGN+98] and [R94].

We also study the more general *heterogeneous postal model* (see [BGN+98]). In this model each vertex v has a delay $0 \leq \rho(v) \leq 1$. The vertex that sends a message is “busy” at the first ρ time units starting from its sending time. In addition, every arc e has a delay $l(e)$ representing the time required to send the message over e . The broadcast problem in the heterogeneous postal model was studied in [BGN+98], where the authors have shown that their algorithm (based on LP) can be adapted to this model, and provides a logarithmic approximation guarantee for the corresponding broadcast problem. We show that our combinatorial algorithm also can be adapted to give an $O(\log n)$ ratio broadcast algorithm for the directed case, and an $O(\log k)$ ratio algorithm for the multicast variant on undirected graphs, within the heterogeneous postal model. The running time of the adapted version of our algorithm is $\tilde{O}(|E||V|)$. In addition, we generalize the heterogeneous postal model and introduce the *arc-dependent heterogeneous postal model*, in which the delay of a vertex v depends on the arc it uses to send the message. We believe that this model captures modern communication networks, in which the major components (links and processors) are not homogeneous, more truthfully than the heterogeneous postal model of [BGN+98] and than all the other models that were previously considered in the literature (such as the *postal model* of [BK94] and the *logP model* of [CKP+96]). We adapt our algorithm to the arc-dependent heterogeneous postal model and show that it provides a logarithmic approximation guarantee for the *directed* and *undirected* versions of the corresponding broadcast problem. However, this most general version of our algorithm

¹We use the notation $\tilde{O}(f(n))$ to denote $O(f(n) \cdot \text{polylog}(f(n)))$.

does use LP and has a slightly higher running time of $\tilde{O}(|V|^3)$. No approximation algorithm was previously known for this version of the problem.

From the point of view of the hardness of approximation, the best known lower bound on the approximation threshold for the (directed and undirected) telephone broadcast problems is $3/2$ [S00]. In this paper we show that it is NP-hard to approximate the undirected broadcast problem within a ratio of $3 - \epsilon$ for any $\epsilon > 0$ and that the directed telephone broadcast problem is $\Omega(\sqrt{\log n})$ -inapproximable unless $NP \subseteq DTIME(n^{O(\log n)})$.

Implications for network design. A large body of research deals with network design problems in which the objective is to optimize more than one optimization criterion simultaneously (see, e.g., [MRR+01]). Such optimization problems are called *bicriteria optimization* problems. Consider the problem of constructing a spanning tree of the input graph that optimizes the maximum degree and the depth of the tree simultaneously. We call this problem the “*degree-depth* problem.” Ravi [R94] has shown that this problem is closely related to the telephone broadcast problem and has designed the first bicriteria approximation algorithm for it that provides simultaneous polylogarithmic approximation to both the maximum degree and the depth. Defining *poise* of a tree to be the sum between its maximum degree and its depth, this result implies directly a polylogarithmic approximation guarantee for the problem of constructing a spanning tree with minimum poise (henceforth, the *poise* problem). The algorithm of [BGN+98] can be adapted to provide a logarithmic bicriteria approximation for the degree-depth problem, implying, consequently, a logarithmic approximation algorithm for the poise problem.

Our algorithm for the telephone broadcast problem can also be adapted to the degree-depth and poise problems, and it provides logarithmic bicriteria approximation for the former and logarithmic approximation for the latter. As in the case with the telephone broadcast problem, our algorithm is the first to apply to the *directed* versions of both problems, and it is the first combinatorial algorithm for these problems. The discussion above about the worst-case running times of the three different approximation algorithms (those of [R94], [BGN+98], and ours) for the telephone broadcast problem is applicable for the degree-depth and poise problems as well.

Consequent developments. After a preliminary version of this paper was published in STOC 2002 [EK02], we published two more papers [EK03a, EK03b] on this subject. In [EK03a] we have shown that the techniques developed in the current paper can be employed to devise a *sublogarithmic* approximation algorithm for the *undirected* versions of the telephone broadcast and multicast problems, as well as for the undirected degree-depth and poise problems. Specifically, the approximation guarantee that is achieved in [EK03a] for the telephone multicast problem is $O(\frac{\log k}{\log \log k})$. In [EK03b] we studied the *directed multicast* problem and devised an approximation algorithm with multiplicative approximation guarantee of $O(\log n)$ and additive approximation guarantee of $O(\sqrt{k})$.

2. Preliminaries.

2.1. Graphs and trees. Let $G = (V, E)$ be a directed graph (henceforth, digraph). Given a subset W of V , let $G(W) = (W, E(W))$, $E(W) = \{\langle u, w \rangle \in E \mid u, w \in W\}$ denote the subgraph *induced* by W .

The distance $dist(u, v)$ between u and v is the number of arcs in the shortest path between u and v . For a pair of subsets A, B of vertices, the distance from A to B is the minimum distance from a vertex $u \in A$ to a vertex $w \in B$.

The outdegree $outdeg(v)$ of a node v is the number of arcs leaving v .

For a vertex $u \in V$, let $N(u) = \{v \mid \langle u, v \rangle \in E\}$ denote the set of *neighbors* of u .

For a digraph $G = (V, E)$, let $G' = (V, E')$, $E' = \{(u, w) \mid \langle u, w \rangle \in E\}$ denote the undirected *underlying graph* of G .

Let T be a digraph whose underlying graph is a tree, i.e., an acyclic and connected graph.

DEFINITION 2.1. *A digraph T as above is called an arborescence if there is a vertex $r \in V$ (called root) such that for any other vertex $v \in V$ there exists a unique directed path from r to v . The vertex r is called the root of the arborescence.*

The depth of an arborescence T , denoted as $h(T)$, is defined by $h(T) = \max_{v \in V'} \text{dist}_T(r, v)$. The degree of an arborescence T , denoted $\text{deg}(T)$, is the maximum out-degree of a vertex in T , i.e., $\max_{v \in V'} \text{outdeg}(v)$. The set of leaves of T , denoted $L(T)$, is the set of the vertices with no outgoing arcs in the directed case and with degree 1 in the undirected case.

A collection of vertex-disjoint arborescences is called *forest*. The *height* (resp., *degree*) of the forest F , denoted $h(F)$ (resp., $\text{deg}(F)$), is defined by $h(F) = \max_{T \in F} h(T)$ (resp., $\text{deg}(F) = \max_{T \in F} \text{deg}(T)$).

2.2. Schedules. At any given moment, the vertices are split into the set of *informed* vertices I and the set of *uninformed* vertices $U = V \setminus I$. At the beginning, $I = \{s\}$. Further, U' denotes the subset of U that contains vertices that still need to be informed; in the broadcast case $U' = U$, while in the multicast case U' may be much smaller.

The message transmission is performed in *rounds*, each requiring one time unit. A round is a matching between the subsets I and U with all the arcs of the matching oriented from a vertex in I to a vertex in U . In other words, in each round vertices that belong to a subset of the set of informed vertices send the message through this matching to a subset of vertices from the set U . Let I_i, U_i denote the subsets of informed and uninformed vertices, respectively, at the beginning of round i . The vertices of U that participate in the round (i.e., belong to the vertex set of the matching) become *informed* and, consequently, are removed from the subset U and are added to the subset I .

A *proper* schedule is a collection of matchings M_1, M_2, M_3, \dots , such that M_i is a matching between the subsets I_i and U_i , and $I = \{s\}$ when the schedule starts and $I = V$ when the schedule ends.

The *length* of a schedule is the number of rounds it uses.

The optimum value. Let opt be the minimum number of rounds required for broadcasting from s on the instance at hand. Since the size of I can at most double in each round, it follows that $\text{opt} \geq \lceil \log n \rceil$. In addition, $\text{opt} \leq n - 1$ since at least one additional vertex becomes informed in each round. (Otherwise, the instance is infeasible.) Our algorithm is provided with a guess opt' of the correct value of opt , and the algorithm distinguishes between the case when no schedule of length at most opt' exists and the case when there exists a schedule of length at most $\text{opt}' \cdot \log n$. This enables us to figure out the correct value of opt up to a logarithmic factor via binary search.

The optimum tree. Consider the optimum schedule. This schedule defines a directed tree denoted by T^* . The parent of u in the tree is the vertex that sent the message to the vertex u . Without loss of generality, it can be assumed that such a vertex is unique.

The following observation is immediate.

Observation 1. $h(T^*), \text{deg}(T^*) \leq \text{opt}$.

Nonlazy schedules. A schedule is called *nonlazy* [BGN+98] if in each round a maximal matching M_i between I and U is used. A schedule of this type uses a simple greedy rule that ensures that the only idle informed vertices will be those that satisfy that each of their neighbors either is in I or is informed in the current round by some other vertex from I .

2.3. Spiders. Spiders were shown to be useful for telephone broadcast in [BGN+98].

A set of directed paths $S = \{P_1, \dots, P_q\}$ all starting at the same vertex v is called a *spider* if the paths are *vertex-disjoint* except for sharing v . The vertex v is called the *head* of the spider.

The *length* of the spider S , denoted $\ell(S)$, is $\max_{P \in S} |P|$.

The *degree* of the spider S , denoted $\deg(S)$, is $|S| = q$, i.e., the number of paths in the spider.

The *value* of the spider S , denoted $\text{val}(S)$, is $\deg(S) + \ell(S) - 1$.

The following observation is immediate.

Observation 2. Using a nonlazy schedule, the head of a spider S can deliver the message to the other spider vertices in $\text{val}(S)$ rounds.

3. Overview of the algorithm. The algorithm is given for the more general problem of multicasting to a subset $\mathcal{T} \subseteq V$ of *terminals*. The analysis of the upper bound is identical, but since performance is evaluated in terms of *opt*, the minimum number of rounds required for broadcast, instead of the minimum number of rounds for multicasting to \mathcal{T} , we obtain logarithmic upper bounds on the approximation thresholds of the directed broadcast problem, the undirected broadcast problem, and the undirected multicast problem but not on the threshold of the directed multicast problem. (In the case of the *undirected* multicast problem, there is an easy way to modify the analysis so that the upper bound would be obtained in terms of the multicast optimum. Analogous modification, however, fails in the case of the *directed multicast* problem.)

An important combinatorial structure that will be used in this paper is called a *fork*. Informally, a fork is a pair of “short” vertex-disjoint paths starting with the same vertex v and ending at two uninformed vertices u, w . A path is short if its length is at most *opt*, the minimum number of rounds required for completing broadcast. The vertex v is called the *head* of the fork. As the paths are short and there are only two paths, it is easy to see that if v is informed, then it can quickly inform both u and w .

Our algorithm iteratively finds as many vertex-disjoint forks as possible. It maintains a set U corresponding to the set of vertices that are uninformed and have not yet been covered by forks, as well as a set U' of uninformed terminals not covered by forks. The set U' is called a *packing* if the graph contains no forks. A useful property that our algorithm utilizes is that vertices of a packing can be informed very efficiently.

We next provide a high-level description of the algorithm.

- *The extraction step:* Extract a maximal collection of vertex-disjoint forks from $G(U)$; i.e., keep extracting until $G(U)$ contains no more forks.
- *The recursion step:* Recursively inform the set of the fork heads.
- *The forking step:* Inform the vertices of the forks, now that the heads have been informed. Since the forks are vertex-disjoint and are short, this can be done in parallel.
- *The packing step:* It remains to inform the vertices of U' that remain in U , which now form a packing. This is done by a reduction to a minimal

maximum-degree set-cover problem in a bipartite graph. The latter graph has the set of informed vertices on one side and the set U' on the other side. The cover is then found by a single computation of maximum flow.

4. Packing and short schedules.

4.1. Packing. In this section, we describe one of the main tools of our algorithm. Let $U' \subseteq U$ be a subset of the uninformed vertices. We say that the vertices of U' are *dispersed*, or form a *packing* in $G(U)$, if for any two vertices $v, u \in U'$, every vertex $w \in U$ is of distance more than opt from either u or v in $G(U)$.

Formally, we have the following definition.

DEFINITION 4.1. *Let $G = (V, E)$ be a digraph, $U' \subseteq U$ be a subset of vertices, and opt be an integer. Then a (U, U') -fork triple is a triple (u, t_1, t_2) with $t_1, t_2 \in U'$ and $u \in U$, such that $u \neq t_2$ and $dist_{G(U)}(u, t_1), dist_{G(U)}(u, t_2) \leq opt$. Also, U' is called a packing with respect to the set U if no (U, U') -fork exists.*

Observe that if no (U, U') -fork triple exists, then for each vertex w in U there may exist at most one descendant in the tree T^* that belongs to the subset U' . If we think of w as “covering” its descendants, then U' is a packing in the sense that each vertex of U' is covered by a different unique vertex of U .

A straightforward way for testing the existence of a (U, U') -fork triple in an n -vertex digraph $G = (V, E)$ is to examine all breadth-first-search (BFS) trees in $G(U)$. Such an examination requires $O(|E||V|)$ time.

4.2. Mentor vertices. Consider a packing U' with respect to the subset U . We assign to each vertex u of U' a *mentor* vertex $m(u)$, which is the *last* vertex along the directed path between s and u in T^* belonging to I . The definition is valid since such a path starts with $s \in I$ and ends at $u \in U$. Observe that the path in T^* from $m(u)$ to u contains only vertices from the subset U (except $m(u)$.) Intuitively, the mentors aid in fast transmission of the message to the vertices of the subset U' .

Let $P^*(m(u), u)$ denote the arcs of the path from $m(u)$ to u in T^* . Then the following lemma holds.

LEMMA 4.2. *The graph induced by the set $\bigcup_{u \in U'} P^*(m(u), u)$ is a collection of vertex-disjoint spiders. The spider heads belong to I , while other spider vertices belong to U . The value of the forest is at most $2 \cdot opt$.*

Proof. By the definition of $m(u)$, all the vertices that belong to the unique path in T^* from $m(u)$ to u , except $m(u)$, belong to the set U . We show that no two paths $P^*(m(z), z)$ and $P^*(m(u), u)$, $z \neq u$, can intersect unless $m(z) = m(u)$. In the latter case, $m(u) = m(z)$ is the only vertex common to the two paths.

For the sake of contradiction, suppose that some $v \in U$ is contained in paths $P^*(m(u), u)$ and $P^*(m(z), z)$. By the definition of a mentor, the path from v to u and the path from v to z both belong to $G(U)$. In addition, the paths $P^*(m(u), u)$, $P^*(m(z), z)$ are of length at most opt (see Observation 1). It follows that the distance to u from v in $G(U)$, and also the distance from v to z in $G(U)$, is at most opt . This contradicts the assumption that U' is a packing.

Finally, note that both the depth and the maximum degree of the tree T^* are at most opt . Hence, $|P^*(m(u), u)| \leq opt$ for every vertex $u \in U'$, and the degree of each spider (which is a subgraph of T^*) is, consequently, at most opt . Hence, the value of this forest of spiders is at most $2 \cdot opt$. \square

It can now be seen that if U' is a packing, not too many vertices can have v as their mentor. Let $\nu(v) = \{u \mid v = m(u)\}$ denote the number of vertices with v as a mentor for $v \in I$.

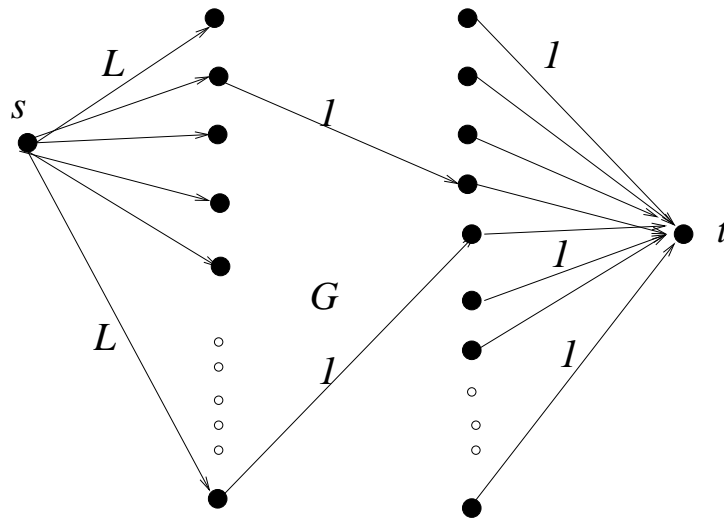


FIG. 1. Computing a cover with minimal maximum degree via maximum flow.

LEMMA 4.3. *If U' is a packing, then $\nu(v) \leq \text{opt}$ for all $v \in I$.*

Proof. Observe that $\nu(v)$ is precisely the degree of the appropriate spider from the forest $\bigcup_{u \in U'} P^*(m(u), u)$. Hence, by Lemma 4.2, $\nu(v) \leq \text{opt}$. \square

4.3. Informing a packing.

Set covers of low degree. Let $B = (V_1, V_2, A)$ be a bipartite graph. We say that $S \subseteq V_1$ covers V_2 if every vertex $v_2 \in V_2$ has a neighbor in S .

DEFINITION 4.4. *We say that a cover S of V_2 has degree d if there exists a mapping $\psi : V_2 \mapsto S$ so that for every $v_2 \in V_2$, $\psi(v_2) \in S \cap N(v_2)$ and so that each vertex in S is assigned at most d vertices of V_2 , namely, for each $v \in S$, $|\{v_2 \in V_2 \mid \psi(v_2) = v\}| \leq d$.*

We next present a reduction from the problem of determining whether there exists a cover for V_2 of degree at most L to the maximum flow problem. Orient the edges of the bipartite graph B from V_1 to V_2 and assign unit capacity to each of them. Add a source s with arcs of capacity L to each vertex of V_1 and a sink t with an arc of unit capacity from each vertex of V_2 . There exists a cover for V_2 of degree at most L if and only if there exists a flow of capacity $|V_2|$. By the integrality of the maximum flow, this flow directly translates to a cover (see Figure 1).

To find a cover of minimum degree, we perform binary search for the value of L . Thus, a cover for V_2 with minimal maximum degree can be found in time $O(\log |V|)T(|E|, |V|)$, where $T(|E|, |V|) = O(|E| \cdot |V| \cdot \log |V|)$ is the time complexity of computing the maximum flow [GT88].

The graph H . Consider the graph $H(I, U', \hat{E})$ that describes all pairs $v \in I$ and $u \in U'$, with v being a potential mentor for u . Formally, we have the following definition.

DEFINITION 4.5. *The bipartite graph $H(I, U', \hat{E})$ has $I \cup U'$ as its vertex set. Two vertices $v \in I$ and $u \in U'$ are connected by an undirected edge in \hat{E} if $\text{dist}_{G(U \cup \{v\})}(v, u) \leq \text{opt}$. In other words, this condition means that there exists a path of length at most opt from v to u that uses only vertices from the set U (except v).*

Low-degree covers for H . Our goal is to show that every packing U' has a low-degree cover in H . Consider the assignment ψ from U' to I defined by $\psi(u) = m(u)$,

where $m(u)$ is the mentor of the vertex u . Note that the arc $\langle u, m(u) \rangle$ belongs to \hat{E} , and thus the assignment ψ defines a cover. In other words, $M = \{m(u) \mid u \in U'\}$ is a subset of I so that each $u \in U'$ has a neighbor in M (where the neighborhood is with respect to the graph H). By Lemma 4.3, $\nu(v) = |\{u \mid \psi(u) = v\}| \leq \text{opt}$. We have proven the following corollary.

COROLLARY 4.6. *There exists a cover of U' in $H(I, U', \hat{E})$ of maximum degree at most opt , and such a cover can be found in time $O(\log |V| \cdot T(|E|, |V|))$, where $T(|E|, |V|) = O(|E| \cdot |V| \cdot \log |V|)$ is the time required for computing a maximum flow.*

Using a low-degree cover to inform U' . Given a cover S of U' with maximum degree at most opt , consider the shortest path $P(\psi(u), u)$ from $\psi(u)$ to u in the graph $G(\{\psi(u)\} \cup U)$. The proof of the following lemma is similar to the one of Lemma 4.2.

LEMMA 4.7. *The union of the paths $P(\psi(u), u)$ induces a collection of vertex-disjoint spiders, whose heads belong to the cover S and whose values are at most $2 \cdot \text{opt}$.*

Proof. The paths $P(\psi(u), u)$ can only intersect at their start vertices, as otherwise a fork triple can be extracted (see the proof of Lemma 4.2). The number of paths in each spider is at most the degree of the cover, which is at most opt . In addition, the length of each path in the spider is at most opt , by the definition of the graph $H(I, U', \hat{E})$. Hence, the value of each spider is at most $2 \cdot \text{opt}$. \square

Since all the spiders have small values, the following procedure informs a packing U' using a small number of rounds.

ALGORITHM 1. *Inform-Packing(I, U, U')*

1. *Construct the graph $H(I, U', \hat{E})$.*
2. *Find an assignment $\psi : U' \mapsto I$ of degree at most opt that covers H using flow computation.*
3. *For each $u \in U$, let $P(\psi(u), u)$ be the shortest path from $\psi(u)$ to u in $G(\{\psi(u)\} \cup U)$.*
4. *Each spider head informs all the vertices that belong to the spider using a nonlazy schedule.*

LEMMA 4.8. *Let \mathcal{P} be the spider collection induced by the union of the paths $P(\psi(u), u)$, $u \in U'$. Any nonlazy schedule broadcasts over \mathcal{P} in at most $2 \cdot \text{opt}$ rounds, informing all the remaining vertices of the set U' .*

Proof. By Lemma 4.7, \mathcal{P} is a union of vertex-disjoint spiders, each of value at most $2 \cdot \text{opt}$. Since the spiders are vertex-disjoint, the lemma now follows from Observation 2. \square

4.4. Transforming U' into a packing: The extraction step. The following algorithm accepts as input a subset U of V and the subset U' and deletes vertices from U , so that for the resulting set U at the end of the extraction, no fork triple is left; U' is a packing. The collection of extracted forks is denoted by \mathcal{F} and their heads by \mathcal{R} .

In the following algorithm, it is assumed for simplicity that when a fork is found from $u \in U$ to t_1, t_2 in U' , the two paths used from u to t_1 and from u to t_2 are found by the BFS procedure (i.e., they are shortest paths). These two paths may intersect.

ALGORITHM 2. *Extract-forks (G, U, U')*

1. $\mathcal{F} \leftarrow \emptyset$.
2. **While** there exists a (U, U') -fork triple F **do**:
 3. Add F into \mathcal{F} .
 4. $U \leftarrow U \setminus V(F)$, $U' \leftarrow U' \setminus V(F)$, where $V(F)$ is the set of vertices in F .
5. **return** (U, U', \mathcal{F}) .

It is easy to see that the resulting forks in \mathcal{F} are vertex-disjoint and that the returned set U' is a packing with respect to the returned set U .

Say that we establish (via a BFS computation) that no two distinct terminals have a short distance from u in $G(U)$. In such a case the vertex u cannot serve as a head of a fork triple. Therefore, the number of BFS explorations is at most $|V|$, and hence extracting the fork triples can be accomplished in time $\tilde{O}(|E| \cdot |V|)$.

5. The algorithm: Combining the pieces. The input to the algorithm is the directed graph G , source vertex s , and a set U' of terminals. The algorithm accepts a guess of opt as part of its input.

5.1. The algorithm. Initially invoked as $ApproximateBroadcast(G, s, V)$, then the algorithm invokes itself recursively with different subsets serving as third parameter. The sequence of these subsets is a monotone decreasing one with respect to containment.

ALGORITHM 3. $ApproximateBroadcast(G(V, E), s, U')$

1. $U \leftarrow V \setminus s, I \leftarrow \{s\}$.
2. /* Transforming U' into a packing */
If U' is not a (U, U') -packing **then**
 - (a) /* The extraction step */
 $Let (U, U', \mathcal{F}) \leftarrow Extract-forks(G, U)$.
 $Let \mathcal{R}$ be the set of heads of \mathcal{F} .
 - (b) /* The recursive step */
 $Recursively\ invoke\ ApproximateBroadcast(G, s, \mathcal{R})$.
 - (c) /* The forking step */
 $Use\ a\ nonlazy\ schedule\ to\ inform\ the\ forks\ in\ \mathcal{F}$.
 $Let\ I \leftarrow I \cup V(\mathcal{F})$.
3. /* The packing step */
 $Invoke\ Inform-Packing(I, U, U')$.

5.2. Analysis of the number of rounds. The following observation follows from the fact that each fork triple contains at most $2 \cdot opt + 1$ vertices.

Observation 3. Each invocation of the forking step (line 2(c) in Algorithm $ApproximateBroadcast$) requires at most $2 \cdot opt$ rounds.

LEMMA 5.1. *The number of rounds used by Algorithm $ApproximateBroadcast$ for broadcasting to an arbitrary set U' is at most $4 \cdot (\log_2(|U'|) + 1) \cdot opt$.*

Proof. We first show that each time lines 2 and 3 of Algorithm $ApproximateBroadcast$ are executed, the number of rounds required is at most $4opt$. The two steps that perform the actual broadcast are the fork-extracting and packing steps. The bound of $4 \cdot opt$ per iteration follows from Lemma 4.8 and Observation 3.

We now analyze the depth of the recursion. Let U'_i (resp., \mathcal{R}_i) be the set of terminals (resp., roots of the forks) on iteration i . Since the forks are vertex-disjoint, and since $|U'_{i+1}| \leq |\mathcal{R}_i|$, the depth of the recursion is at most $\log_2 |U'| + 1$. Hence, overall, the number of rounds of the constructed broadcast is at most $4 \cdot (\log_2(|U'|) + 1) \cdot opt$. \square

To summarize, we have the following theorem.

THEOREM 5.2. *Algorithm $ApproximateBroadcast$ is an $O(\log n)$ -approximation algorithm for the directed telephone broadcast problem.*

Note that the main obstacle to generalizing Algorithm $ApproximateBroadcast$ to the multicast problem is the fact that the set \mathcal{R} is not necessarily contained in the set U' . In the undirected case this obstacle can be overcome by using *fork pairs* instead of fork triples. Intuitively, a fork pair is a pair of nearby terminals of U' that

are still not used for other fork pairs. From each fork pair, precisely one terminal is inserted to \mathcal{R} , guaranteeing both $|\mathcal{R}| \leq |U'|/2$ and $\mathcal{R} \subseteq U'$. This results in an $O(\log k)$ -approximation for the *undirected k -multicast problem*.

5.3. The running time of the algorithm. The most time-consuming computational tasks in each iteration are extractions of forks and computations of maximum flow. Recall that the extraction step can be completed with at most $|V|$ BFS computations, in time $O(|V||E|)$, while the time for finding a low-degree cover is $O(|V||E|\log|V|)$ [GT88]. Since each iteration reduces the number of target terminals by a factor of at least 2, the depth of the recursion is $O(\log n)$. Hence, the time complexity of the algorithm is $O(|V||E|\log^2|V|)$. Finding the optimal value of opt via binary search adds one more logarithmic factor.

5.4. The postal and arc-dependent postal models. In this section we consider some generalized versions of telephone communication.

The heterogeneous postal model. We start with the *heterogeneous postal model*, introduced in [BGN+98], in which each vertex v has a delay $0 \leq \rho(v) \leq 1$. The vertex that sends a message is “busy” at the first ρ time units starting from its sending time. In addition, every arc e has a delay $l(e)$ representing the time required to send the message over e . (Note that in the context of these generalized models one has to consider a continuous time scale instead of a discrete one, and hence the notion of “round” becomes obsolete.)

To adapt Algorithm ApproximateBroadcast to the heterogeneous postal model, the paths in forks should be weighted (with weight function determined by the delay function l). In addition, when computing a low-degree cover of U' , the delay $\rho(v)$ for $v \in I$ has to be taken into account. A vertex v can have up to $\lfloor opt/\rho(v) \rfloor$ children in the tree T^* (that is, possibly more than opt). Hence, the capacities of the arcs that are adjacent to the source s_1 in the flow graph $H(I, U', \hat{E})$ (see section 4.3) should be set as $opt/\rho(v)$. Finally, when constructing the arc set \hat{E} , the distances between $v \in I$ and $u \in U'$ in the graph $G(\{v\} \cup U)$ should take into account the length of the arcs (i.e., they are weighted distances).

The rest of the proof can be carried out in an analogous way. Particularly, the cover with bounded degree gives rise to a collection of “narrow” spiders. A vertex v that heads one of these spiders and has degree at most $opt/\rho(v)$ in the spider can deliver the message to all its children in the spider in at most opt time units.

The arc-dependent postal model. We next generalize our algorithm even further. Consider the *arc-dependent heterogeneous postal model*, in which the delay of a vertex v depends on the arc through which it chooses to send the message.

Consider again the construction of the flow graph $H(I, U', \hat{E})$. Observe that despite the assumption that the set U' is a packing, even in the standard telephone model for a given pair of vertices $v \in I$, $u \in U'$, there might be more than one shortest path from v to u in the graph $G(U \cup \{v\})$. Let $N(v)$ denote the set of neighbors of v . For each vertex $z \in N(v)$, let P_z denote the shortest path from v to u that passes through the vertex z . Let $\omega(P_z) = \sum_{e \in P_z} l(e)$ denote the *weight* of the path P_z . The edge (v, u) is added to the flow graph $H(I, U', \hat{E})$ if for some $z \in N(v)$, the weight of the path P_z plus the delay $\rho(\langle v, z \rangle)$ incurred by the vertex v while sending a message through the arc $\langle v, z \rangle$ is at most opt . (Observe that this is a direct generalization of the construction for the heterogeneous postal model.)

Let $C_u(v) \subseteq N(v)$ be the subset of $N(v)$ that contains only those neighbors z of $N(v)$ for which $\omega(P_z) + \rho(\langle v, z \rangle) \leq opt$. Suppose that for every vertex $u \in U'$ that is connected to v in \hat{E} we would pick an arbitrary vertex $z_u \in C_u(v)$ for serving as a relay

station for broadcasting the message from v to u . (This is exactly what is implicitly done in the telephone and heterogeneous postal models.) The weighted degree of the vertex v in the spider $\bigcup\{P_{z_u} \mid u \in U', (v, u) \in \hat{E}\}$ would be $\deg(v) = \sum\{\rho(\langle v, z_u \rangle) \mid u \in U', (v, u) \in \hat{E}\}$. Note that unlike the telephone and heterogeneous postal models, in the arc-dependent model $\deg(v)$ depends on the choice of the vertices z_u . To tackle this difficulty, we assign to each edge $(v, u) \in \hat{E}$ capacity $\rho(\langle v, z_u \rangle)$, where z_u is the vertex of $C_u(v)$ that minimizes $\rho(\langle v, z_u \rangle)$. We need to choose a cover $S \subseteq I$ and an assignment $\psi : U' \rightarrow I$ that minimizes $\max_{v \in S}\{\deg_\psi(v)\}$, where $\deg_\psi(v)$ is given by $\sum\{\rho(\langle v, z_u \rangle) \mid u \text{ such that } \psi(u) = v\}$.

We claim that the set $S = \{\psi(u) \mid u \in U'\}$, for $\psi(u) = m(u)$, is a cover of U' of weighted degree at most opt . Indeed, let $v = m(u)$. Consider the path $P^*(v, u)$ from v to u in T^* . Let w be the neighbor of v in this path.

Note that $w \in C_u(v)$. Hence the delay in sending the message from v to w is $\rho(v, w) \geq \rho(v, z_u)$. Therefore, the weighted degree of the assignment defined by T^* is a lower bound on the optimum broadcast time. Thus, this weighted degree cannot be larger than opt . It follows that the assignment $\psi(u) = m(u)$ yields a cover assignment for U' with maximum weighted degree at most opt .

We want to find a cover S and an assignment ψ that minimizes (or almost minimizes) $\max_{v \in S}\{\deg_\psi(v)\}$. The problem of finding the best weighted degree assignment is known in the literature as the problem of *scheduling of independent parallel machines*. This problem is NP -hard and, in fact, is $3/2$ -inapproximable [LST90].

For our purposes, any constant approximation for this problem is sufficient. Unfortunately, we are not aware of a combinatorial algorithm that provides a constant approximation ratio for this problem. We use the 2-approximation algorithm of [LST90] that formulates the problem as an integer linear program, relaxes it to allow fractional solutions, finds a basic feasible solution, and uses a standard rounding technique.

Hence, this way we find a (weighted) cover for the set U' of degree at most $2 \cdot \text{opt}$. It follows that the value of the resulting spiders is at most $3 \cdot \text{opt}$. The rest of the analysis can be carried through in a straightforward way. To summarize, our algorithm can be adapted to provide a logarithmic approximation guarantee for the directed and undirected versions of broadcast problem in the arc-dependent heterogeneous postal model. The approximation guarantee of the generalized algorithm is only by a constant factor greater than the approximation guarantee of our algorithm for the telephone broadcast problem; i.e., it is $O(\log n)$. Note, however, that for this most general version of our algorithm we do use LP. The particular linear programs that are used in the algorithm are solvable via Lagrangian relaxation in time $\tilde{O}(|V|^3)$ [LST90], and so the overall running time of the generalized algorithm is at most $\tilde{O}(|V|^3)$ as well.

5.5. Implications for network design.

A bicriteria approximation for depth and outdegree. Our algorithm provides a bicriteria approximation for the depth-degree problem. In other words, given a digraph for which there exists a spanning arborescence of height h and maximum degree d , our algorithm constructs a spanning arborescence of maximum depth $O(\log n) \cdot h$ and maximum degree $O(\log n) \cdot d$.

The tree is built “backwards” (from the leaves to the root). Throughout the algorithm we maintain a forest. The number of trees in the forest gradually decreases until they merge into a single spanning arborescence.

We show how to build this arborescence recursively. The collection of forks \mathcal{F} that is computed by our algorithm on the first level of the recursion forms the initial forest. The heads \mathcal{R} need now to be recursively connected to the root s . Let T' be the

arborescence connecting s to \mathcal{R} that was computed by the recursion. By the inductive hypothesis, the set T' is indeed an arborescence, and, in particular, each arc appears in T' at most once. However, observe that T' may not be arc- or vertex-disjoint with the forest \mathcal{F} . We need to unite T' and \mathcal{F} in order get a single spanning arborescence. This is done by taking the graph induced by $\mathcal{F} \cup T'$ and computing the shortest path arborescence from s to all the vertices $v \neq s$ in $G(\mathcal{F} \cup T')$.

Clearly, the resulting shortest path arborescence T'' has depth $O(\log n) \cdot h$, because each recursive iteration can add at most h to the depth of the arborescence. Similarly, on each recursive invocation, for every vertex v at most d arcs that are adjacent to v are added to the arborescence. Hence, the upper bound of $O(\log n) \cdot d$ on the maximum degree follows.

Hence, our algorithm provides an $(O(\log n), O(\log n))$ -bicriteria approximation for both the directed and undirected versions of the degree-depth problem. The algorithm is combinatorial, and its running time is $\tilde{O}(|E| \cdot |V|)$. Note that the same algorithm provides a logarithmic approximation for the directed and undirected poise problems.

6. Hardness results. In this section we present some lower bounds on the approximation thresholds of the (undirected and directed) telephone broadcast problems.

DEFINITION 6.1. *Given an undirected bipartite graph $G = (V_1, V_2, E)$, a subset of vertices $S \subseteq V_1$ is a set cover for G if for any $v_2 \in V_2$ there exists $v_1 \in S$ such that $(v_1, v_2) \in E$.*

Let $V = V_1 \cup V_2$. Let $|V| = n$.

Given a constant $0 < c \leq 1$ and an integer-valued function $t = t(n)$, the YES-instance of the set-cover $(t(n), c)$ -promise problem is an undirected bipartite graph G for which there exists a set cover S of size $|S| \leq t(n)$. The NO-instance of the set-cover $(t(n), c)$ -promise problem is an undirected bipartite graph G for which any set cover S has size $|S| \geq c \log n \cdot t(n)$.

The set-cover $(t(n), c)$ -promise problem is given either a YES-instance or a NO-instance of the problem, and the goal is to determine whether it is a YES-instance or a NO-instance.

It is known [LY95] that there exists a constant c such that the set-cover $(t(n), c)$ -promise problem is NP-hard for $t(n) = \lceil \sqrt{n} \rceil$.

Given an instance of the set-cover $(t(n), c)$ -promise problem, an undirected bipartite n -vertex graph $G = (V_1, V_2, E)$, $|V_1| = |V_2| = n/2$, we construct a graph \bar{G} in the following way. Insert into \bar{G} an isomorphic copy of G . For $j = 1, 2$, let \bar{V}_j be the image of V_j under the isomorphism, and, more generally, for a subset $X \subseteq V$ of vertices, let \bar{X} be the image of X under the isomorphism. Add a new vertex s and connect it with all the vertices of \bar{V}_1 via outgoing arcs. In addition, add a directed path of length $\lceil (c/2)t(n) \log n \rceil$ that starts in s and contains $\lceil (c/2)t(n) \log n \rceil$ new vertices. Denote by s' the tail of the path. Construct a complete binary arborescence T of depth $\lceil \log n \rceil$, rooted at s' , with the set of leaves that contains \bar{V}_1 and (maybe) some new vertices, and such that all the remaining vertices of T are new vertices. Finally, replace each star $(\bar{V}_1, \bar{V}_2^1), (\bar{V}_1, \bar{V}_2^2), \dots, (\bar{V}_1, \bar{V}_2^r)$, with $\bar{V}_1 \in \bar{V}_1, \bar{V}_2^1, \bar{V}_2^2, \dots, \bar{V}_2^r \in \bar{V}_2$, by a complete binary arborescence of depth $\lceil \log r \rceil$ rooted at \bar{V}_1 and whose set of leaves $L(T)$ contains the set $\{\bar{V}_2^1, \bar{V}_2^2, \dots, \bar{V}_2^r\}$ and (maybe) some new vertices. All the other vertices of the tree T are new vertices.

This completes the description of \bar{G} (see Figure 2). Its construction is very similar to the reduction of [S00] that proves the hardness of $3/2$ for the problem. We next use \bar{G} as a building block in our reduction that shows the hardness of $\Omega(\sqrt{\log n})$ for

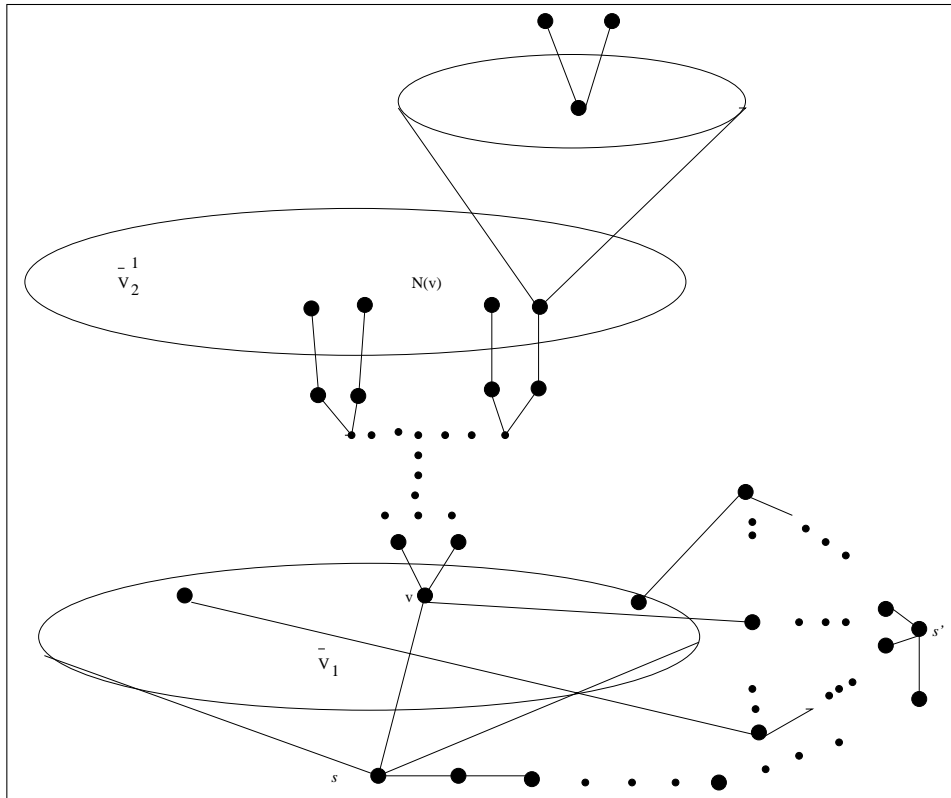


FIG. 2. The first level of the graph \bar{G}^i . The path from s to s' is of length $\Theta(t(n) \cdot \log n)$. The vertex s' is connected via a complete binary arborescence to the vertex set \bar{V}_1 . The vertices of \bar{V}_1 are connected via complete binary arborescences to their neighbors in \bar{V}_2 . In turn, the vertices of the set \bar{V}_2 serve as sources of the next level of the recursive construction.

the directed broadcast problem and then, via a similar reduction, a hardness of $3 - \epsilon$ for any $\epsilon > 0$ for the undirected broadcast problem.

We first analyze the properties of the graph \bar{G} .

LEMMA 6.2. *If G is a YES-instance of the set-cover $(t(n), c)$ -promise problem, then there exists an $(\{s\}, \bar{V}_2)$ -schedule Π of length $|\Pi| = O(t(n))$ for the instance (\bar{G}, s) of the directed telephone broadcast problem.*

Proof. We describe the schedule Π that satisfies the assertion of the lemma. Let $S \subseteq V_1$ be an optimal set cover of the instance G . The schedule starts with delivering the message from the vertex s to all the vertices of the set \bar{S} . Note that as s is connected via outgoing arcs to all the vertices of \bar{V}_1 , it follows that all the vertices of the set \bar{S} will be informed within $|S|$ rounds. By assumption, G is a YES-instance of the set-cover $(t(n), c)$ -promise problem, and hence $|S| \leq t(n)$. Hence, this step requires at most $t(n)$ rounds.

Next, the schedule delivers the message from \bar{S} to all the vertices of the set \bar{V}_2 . Observe that since the set S is a set cover for the set V_2 , only $O(\log n)$ rounds will be required to deliver the message to all the vertices of \bar{V}_2 through the auxiliary arborescences that connect the sets \bar{V}_1 and \bar{V}_2 . Hence, altogether, the length of this schedule is at most $t(n) + \log n = O(t(n))$ (since $t(n) = \lceil \sqrt{n} \rceil$), and it informs all the vertices of the set \bar{V}_2 . \square

LEMMA 6.3. *If G is a NO-instance of the set-cover $(t(n), c)$ -promise problem, then any $(\{s\}, \bar{V}_2)$ -schedule Π for the instance (\bar{G}, s) of the directed telephone broadcast problem has length $|\Pi| = \Omega(t(n) \log n)$.*

Proof. By construction, the only way to inform the vertices of the set \bar{V}_2 is to inform all the vertices of a subset $\bar{S} \subseteq \bar{V}_1$ for some set cover S of V_1 .

There are two possible ways to inform such a subset \bar{S} : either using the arcs that connect s to the vertices of \bar{V}_1 , or through the path between s and s' , and via the complete binary arborescence T (or by combining these two ways).

Note that as G is a NO-instance, it follows that $|S| = |\bar{S}| = \Omega(t(n) \cdot \log n)$. Therefore, the first way requires $\Omega(t(n) \cdot \log n)$ rounds of telephone broadcast. Also, recall that the length of the path from s to s' is $\Omega(t(n) \cdot \log n)$, and hence informing even a single vertex $v \in \bar{V}_1$ using the second way would require $\Omega(t(n) \cdot \log n)$ rounds. Hence, any schedule that informs a subset $\bar{S} \subseteq \bar{V}_1$ of size $\Omega(t(n) \cdot \log n)$ requires $\Omega(t(n) \cdot \log n)$ rounds of telephone broadcast, proving the claim. \square

Note that Lemmata 6.2 and 6.3 imply $\Omega(\log n)$ -inapproximability of the *directed multicast problem*, that was known [F01].

However, since for both the YES and NO-instance of the set cover, informing \bar{V}_1 requires $\Omega(t(n) \log n)$ rounds, this reduction by itself may provide only a constant hardness of approximation for the *broadcast problems*. Indeed, by a careful choice of parameters and some modifications to \bar{G} , it is shown in [S00] that this reduction yields a hardness of $3/2$ for the undirected broadcast problem.

Consider the following recursive construction of a triple $(\bar{G}^i = (\bar{V}^i, \bar{E}^i), s^i, \bar{V}_2^i)$, where (\bar{G}^i, s^i) is an instance of the directed broadcast problem, and $\bar{V}_2^i \subseteq \bar{V}^i$. Let $(\bar{G}^1, s^1, \bar{V}_2^1) = (\bar{G}, s, \bar{V}_2)$. Let also $Z(\bar{G}^1)$ denote the set \bar{V}_2^1 . Given a construction of $(\bar{G}^i, s^i, \bar{V}_2^i)$, the triple $(\bar{G}^{i+1}, s^{i+1}, \bar{V}_2^{i+1})$ is constructed in the following way.

Insert an isomorphic copy of \bar{G}^1 into \bar{G}^{i+1} . Set $s^{i+1} = s^1$. For $j = 1, 2$, let \bar{V}_j^1 be an image of \bar{V}_j under the isomorphism. Insert the set \bar{V}_2^1 into \bar{V}_2^{i+1} . For every vertex $\bar{V}_2^1 \in \bar{V}_2^1$, construct a triple $(\bar{G}^i(\bar{V}_2^1), \bar{V}_2^1, \bar{V}_2^i(\bar{V}_2^1))$, where $\bar{G}^i(\bar{V}_2^1)$ is an isomorphic copy of \bar{G}^i such that the isomorphism takes s^i to \bar{V}_2^1 . All the other vertices of $\bar{G}^i(\bar{V}_2^1)$ are new vertices. Insert the set $\bar{V}_2^i(\bar{V}_2^1)$ into \bar{V}_2^{i+1} . Form the set $Z(\bar{G}^{i+1})$ to be the union of the isomorphic copies of the sets $Z(\bar{G}^i(\bar{V}_2^1))$ for all the different vertices $\bar{V}_2^1 \in \bar{V}_2$.

This completes the construction of $(\bar{G}^{i+1} = (\bar{V}^{i+1}, \bar{E}^{i+1}), s^{i+1}, \bar{V}_2^{i+1})$.

First, we provide an estimate of the number of vertices of the graph \bar{G}^i .

LEMMA 6.4. $|\bar{V}^i| = n^{O(i)}$, $i = 1, 2, \dots$

Proof. The proof follows by a straightforward induction on the number of levels i . \square

Next, we analyze the graph \bar{G}^i that is obtained by the reduction that was described above from a YES-instance G of the set-cover $(t(n), c)$ -promise problem.

LEMMA 6.5. *If G is a YES-instance of the set-cover $(t(n), c)$ -promise problem, then for any $i = 1, 2, \dots$, there exists an $(\{s^i\}, \bar{V}_2^i)$ -schedule Π for the instance (\bar{G}^i, s^i) of the directed broadcast problem of length $|\Pi| = O(i \cdot t(n))$.*

Proof. The proof is by induction on i . The induction base follows from Lemma 6.2.

For the induction step consider the instance (\bar{G}^{i+1}, s^{i+1}) . Note that by Lemma 6.2, all the vertices of the set \bar{V}_2^1 can be informed within $O(t(n))$ rounds. After that point every vertex $\bar{V}_2^1 \in \bar{V}_2^1$ needs to relay the message to all the vertices of the set $\bar{V}_2^i(\bar{V}_2^1)$ through an isomorphic copy of the graph \bar{G}^i . Since the different copies share no vertices, these broadcasts can be conducted in parallel and, by the induction hypothesis, can be completed within $O(i \cdot t(n))$ rounds. \square

LEMMA 6.6. *For any $i = 1, 2, \dots$, there exists a $(\{s^i\} \cup \bar{V}_2^i, \bar{V}^i \setminus \bar{V}_2^i)$ -schedule Π of length $O(t(n) \cdot \log n)$.*

Proof. Consider the graph \bar{G}^i , and suppose that for every isomorphic copy of the graph \bar{G} that is contained in \bar{G}^i , the vertex that corresponds to s is informed.

Consider some single copy of the graph \bar{G} . Observe that delivering the message from s to all the vertices of the set \bar{V}_1 through the path from s to s' and from s' through the complete binary arborescence to all the vertices of the set \bar{V}_1 requires only $O(t(n) \cdot \log n)$ rounds. This is because the distance between s and s' is $O(t(n) \cdot \log n)$, and after s' gets the message only $O(\log n)$ additional rounds are required to relay the message from s' to the vertices of the set \bar{V}_1 .

Since once \bar{V}_1 are informed only $O(\log n)$ more rounds are required to inform \bar{V}_2 , altogether $O(t(n) \cdot \log n)$ rounds are required to deliver the message from s to all the other vertices of the copy \bar{G} , and, furthermore, these deliveries can be conducted in parallel in different copies. \square

COROLLARY 6.7. *If G is a YES-instance of the set-cover $(t(n), c)$ -promise problem, then for any $i = 1, 2, \dots$, there exists an $(\{s^i\}, \bar{V}^i)$ -schedule of length $O(t(n) \cdot (i + \log n))$.*

Next, we turn to analyzing the graph \bar{G}^i that is obtained by the reduction that was described above from a NO-instance G of the set-cover $(t(n), c)$ -promise problem.

LEMMA 6.8. *If G is a NO-instance of the set-cover $(t(n), c)$ -promise problem, then for any $i = 1, 2, \dots$, any $(\{s^i\}, \bar{V}^i)$ -schedule Π for the instance (\bar{G}^i, s^i) of the directed broadcast problem is of length $|\Pi| = \Omega(t(n) \cdot \log n \cdot i)$.*

Proof. The proof is by induction on i . The induction base follows directly from Lemma 6.3.

For the induction step, consider the graph \bar{G}^{i+1} , and consider the isomorphic copy of \bar{G}^1 that has the vertex s^{i+1} as the image of the vertex s^1 under the isomorphism. For every schedule Π , by Lemma 6.3, there exists a vertex $\bar{V}_2^1 \in \bar{V}_2^1$ in this copy that is informed only after $\Omega(t(n) \cdot \log n)$ rounds. Consider the copy $\bar{G}^i(\bar{V}_2^1)$ of the graph \bar{G}^i . Recall that by construction, this copy of \bar{G}^i is a subgraph of the graph \bar{G}^{i+1} . Also, no vertex of this copy $\bar{G}^i(\bar{V}_2^1)$ can be informed before the vertex \bar{V}_2^1 is informed. By the induction hypothesis, delivering the message from the vertex \bar{V}_2^1 to all the other vertices of the copy $\bar{G}^i(\bar{V}_2^1)$ requires $\Omega(i \cdot t(n) \cdot \log n)$ rounds. The assertion of the lemma follows. \square

Substituting $i = \log n$ into Corollary 6.7 we get a broadcast time of $O(\log n) \cdot t(n)$ for a YES-instance. By substituting $i = \log n$ into Lemma 6.8 we get a broadcast time of $\Omega(\log^2 n \cdot t(n))$ for a NO-instance. Hence the gap is $\Theta(\log n)$.

We now compare the gap to the number of vertices in the broadcast instance. For $i = \log n$, the number N of vertices in the broadcast instance is $N = n^{\log n}$. Hence, $\log n = \sqrt{\log N}$. Hence, we get a gap which equals the square of the log of the size of the instance.

THEOREM 6.9. *The directed broadcast problem is $\Omega(\sqrt{\log n})$ -inapproximable unless $NP \subseteq DTIME(n^{O(\log n)})$.*

Next, we establish inapproximability of $3 - \epsilon$ for any $\epsilon > 0$ for the undirected broadcast problem. This is done via a similar reduction.

Specifically, the graph \bar{G} is modified in the following way. The distance between s and s' vertices is changed to $c \cdot t(n) \cdot \log n$. (In the directed construction it was half this value.) Insert into \bar{G} an isomorphic copy of \bar{G} , with all the directed arcs replaced by undirected edges. For $j = 1, 2$ let \bar{V}_j be the image of \bar{V}_j under the isomorphism, and, more generally, for a subset $X \subseteq V$, let \bar{X} denote its image under

the isomorphism. Second, for every star $(v_1, v_2^1), (v_1, v_2^2), \dots, (v_1, v_2^r)$ in G with $v_1 \in V_1, v_2^1, v_2^2, \dots, v_2^r \in V_2$ (or, in other words, for every complete binary tree rooted at \bar{V}_1 and with the set of leaves that contains the set $\{\bar{V}_2^1, \dots, \bar{V}_2^r\}$ in \bar{G}), insert a path of length $\lceil \log n \cdot c/2 \cdot t(n) \rceil$ that connects \bar{V}_1 with a vertex \bar{V}'_1 , where all the vertices of this path except of \bar{V}_1 are new. Now construct a complete binary tree rooted in \bar{V}'_1 and with the set of leaves that contains $\{\bar{V}_2^1, \dots, \bar{V}_2^r\}$, and all the other vertices of the tree are new. This completes the construction of \bar{G} .

Now the construction of the graph \bar{G}^i for $i = 1, 2, \dots$, given the graph \bar{G} , is identical to the construction of \bar{G}^i out of \bar{G} .

LEMMA 6.10. *If G is a YES-instance of the set-cover $(t(n), c)$ -promise problem, then there exists an $(\{s\}, \bar{V}_2)$ -schedule Π of length $|\Pi| \leq (c/2 + o(1))t(n) \log n$ for the instance (\bar{G}, s) of the undirected telephone broadcast problem.*

Proof. As G is a YES-instance of the set-cover problem, there exists a set cover $S \subseteq V_1$ for V_2 of size at most $t(n)$. Using the edges of the star that connect between the copy of the vertex s in \bar{G} and the copies of the vertices of S , it is possible to inform the latter vertices using $|S| \leq t(n)$ rounds. Since S is a set cover for V_2 , and since the distance between the sets \bar{S} and \bar{V}_2 in the graph \bar{G} is $\frac{c}{2} \cdot \log n \cdot t(n) + O(\log n)$, it follows that it is possible to relay the message from the vertices of the set \bar{S} to all the vertices of the set \bar{V}_2 in $(\frac{c}{2} + o(1)) \cdot \log n \cdot t(n)$ rounds. \square

LEMMA 6.11. *If G is a NO-instance of the set-cover $(t(n), c)$ -promise problem, then any $(\{s\}, \bar{V}_2)$ -schedule Π for the instance (\bar{G}, s) of the undirected telephone broadcast problem has length $|\Pi| \geq 3/2 \cdot c \cdot t(n) \log n$.*

Proof. The proof of this lemma is analogous to that of Lemma 6.3. As in that proof, it is easy to see that for informing all the vertices of the set \bar{V}_2 it is necessary to inform a subset $\bar{S} \subseteq \bar{V}_1$ that satisfies that the subset S is a set cover for V_2 . Informing the set \bar{S} via the edges that connect the vertex s to the vertices of \bar{V}_1 requires at least $|\bar{S}| \geq c \cdot t(n) \cdot \log n$ rounds (because G is a NO-instance). Informing a single vertex from the set \bar{V}_2 using the path that connects s to s' and using the complete binary tree T rooted in s' requires at least $c \cdot t(n) \cdot \log n$ rounds, because the length of this path between s and s' is $c \cdot t(n) \cdot \log n$. Hence, at least $c \cdot t(n) \cdot \log n$ rounds are required to inform a set \bar{S} , where S is a set cover for V_2 . Finally, since the distance between the sets \bar{V}_1 and \bar{V}_2 in the graph \bar{G} is at least $\frac{c}{2} \cdot t(n) \cdot \log n$, it follows that informing the vertices of the set \bar{V}_2 once all the vertices of the subset \bar{S} are informed would require at least $\frac{c}{2} \cdot t(n) \cdot \log n$ additional rounds. Hence, altogether, at least $\frac{3}{2}c \cdot t(n) \cdot \log n$ rounds are required to relay the message from s to all the vertices of the set \bar{V}_2 . \square

Note that Lemmata 6.10 and 6.11 imply that $(3 - o(1))$ -inapproximability of the undirected multicast problem with the multicast task is informing V_2 . However, since for both the YES and NO-instance of the set-cover problem, informing the subset \bar{V}_1 in the instance (\bar{G}, s) of the undirected telephone broadcast problem requires $t(n) \cdot \log n(c + o(1))$ rounds, the reduction that uses only \bar{G} yields only $(3/2 - o(1))$ -inapproximability for the undirected broadcast problem. This is, essentially, the reduction of [S00].

In the following lemmata we show that the construction of \bar{G}^i yields the desired $(3 - \epsilon)$ -inapproximability.

LEMMA 6.12. $|\bar{V}^i| = n^{O(i)}, i = 1, 2, \dots$

Proof. Similarly to Lemma 6.4, the proof can be derived by a straightforward induction on i . \square

The proof of the following lemma is analogous to the proofs of Lemmata 6.5 and 6.6.

LEMMA 6.13.

1. If G is a YES-instance of the set-cover $(t(n), c)$ -promise problem, then for any $i = 1, 2, \dots$, there exists an $(\{s^i\}, \bar{V}_2^i)$ -schedule Π for the instance (\bar{G}^i, s^i) of the undirected broadcast problem of length $|\Pi| \leq i \cdot t(n)(c/2 + o(1)) \log n$.
2. For any $i = 1, 2, \dots$, there exists a $(\{s^i\} \cup \bar{V}_2^i, \bar{V} \setminus \bar{V}_2^i)$ -schedule Π of length $|\Pi| \leq t(n) \log n(c/2 + o(1))$.

The following corollary is immediate, given Lemma 6.13.

COROLLARY 6.14. If G is a YES-instance of the set-cover $(t(n), c)$ -promise problem, then for any $i = 1, 2, \dots$, there exists an $(\{s^i\}, \bar{V}^i)$ -schedule Π for the instance (\bar{G}^i, s^i) of the undirected broadcast problem of length $|\Pi| \leq (i+1)t(n) \log n(c/2 + o(1))$.

We next turn to the analysis of the NO-instance.

LEMMA 6.15. If G is a NO-instance of the set-cover $(t(n), c)$ -promise problem, for any $i = 1, 2, \dots$, any $(\{s^i\}, \bar{V}^i)$ -schedule for the instance (\bar{G}^i, s^i) of the undirected broadcast problem is of length $|\Pi| \geq 3 \cdot t(n) \log n(c/2 + o(1)) \cdot i$.

Proof. The proof of this lemma is analogous to that of Lemma 6.8. The only difference is that Lemma 6.11 is used instead of Lemma 6.3. \square

It follows that the undirected broadcast problem is $(3 - O(1/i))$ -inapproximable unless $NP \subseteq DTIME(n^{O(i)})$.

THEOREM 6.16. For any $\epsilon > 0$, it is NP-hard to approximate the undirected broadcast problem within a ratio of $3 - \epsilon$.

7. Discussion. We demonstrated that the approximation threshold of the directed telephone broadcast problem is between $O(\log n)$ and $\Omega(\sqrt{\log n})$. Bridging this gap is a challenging open problem. Determining the approximation threshold of the directed multicast problem is another important open problem. Here the gap is between an additive approximation of $O(\sqrt{k})$ (that comes together with a logarithmic multiplicative factor) [EK03b] and a lower bound of $\Omega(\log n)$ [F01].

Acknowledgments. We are grateful to Magnus M. Halldorsson for his help in writing this manuscript and for helpful discussions and comments. We are also grateful to Zvika Lotker for helpful and motivating discussions, particularly about the reduction of [S00]. Finally, we thank two anonymous referees for helpful comments.

REFERENCES

- [BGN+98] A. BAR-NOY, S. GUHA, J. NAOR, AND B. SCHIEBER, *Message multicasting in heterogeneous networks*, SIAM J. Comput., 30 (2000), pp. 347–358.
- [BK94] A. BAR-NOY AND S. KIPNIS, *Designing broadcasting algorithms in the postal model for message-passing systems*, Math. Systems Theory, 27 (1994), pp. 431–452.
- [CKP+96] D. E. CULLER, R. M. KARP, D. A. PATTERSON, A. SAHAY, E. E. SANTOS, K. E. SCHAUER, R. SUBRAMONIAN, AND T. VON-EICKEN, *LogP: A practical model of parallel computation*, Commun. ACM, 39 (1996), pp. 78–85.
- [EK02] M. ELKIN AND G. KORTSARZ, *A combinatorial logarithmic approximation algorithm for the directed telephone broadcast problem*, in Proceedings of the 34th Annual ACM Symposium on Theory of Computing (STOC 2002), 2002, pp. 438–447.
- [EK03a] M. ELKIN AND G. KORTSARZ, *A sublogarithmic approximation algorithm for the undirected telephone broadcast problem: A path out of a jungle*, in Proceedings of the Fourteenth Annual Symposium on Discrete Algorithms ACM-SIAM (SODA'03), 2003, pp. 76–85.

- [EK03b] M. ELKIN AND G. KORTSARZ, *An approximation algorithm for the directed telephone multicast problem*, in Proceedings of the International Colloquium on Automata, Languages, and Programming (ICALP 2003), pp. 212–223.
- [F01] P. FRAIGNAUD, *Approximation algorithms for minimum-time broadcast under the vertex-disjoint paths mode*, in Proceedings of the 9th Annual European Symposium on Algorithms (ESA '01), 2001, pp. 440–451.
- [GT88] A. V. GOLDBERG AND R. E. TARJAN, *A new approach to the maximum flow problem*, J. Assoc. Comput. Mach., 35 (1988), pp. 921–940.
- [HHL88] S. M. HEDETNIEMI, S. T. HEDETNIEMI, AND A. LIESTMAN, *A survey of broadcasting and gossiping in communication networks*, Networks, 18 (1988), pp. 319–349.
- [KP95] G. KORTSARZ AND D. PELEG, *Approximation algorithms for minimum-time broadcast*, SIAM J. Discrete Math., 8 (1995), pp. 401–427.
- [LST90] L. K. LENSTRA, D. SHMOYS, AND E. TARDOS, *Approximation algorithms for scheduling unrelated parallel machines*, Math. Programming, 46 (1990), pp. 259–271.
- [LY95] C. LUND AND M. YANNAKAKIS, *On the hardness of approximating minimization problems*, J. Assoc. Comput. Mach., 41 (1994), pp. 960–981.
- [MRR+01] M. V. MARATHE, R. RAVI, S. S. RAVI, D. J. ROSENKRANTZ, AND H. B. HUNT, *Approximation algorithms for degree-constrained minimum-cost network-design problems*, Algorithmica, 31 (2001), pp. 58–78.
- [R94] R. RAVI, *Rapid rumor ramification: Approximating the minimum broadcast time*, in Proceedings of the 35th Annual IEEE Symposium on Foundations of Computer Science (FOCS '94), 1994, pp. 202–213.
- [S00] C. SCHINDELHAUER, *On the inapproximability of broadcasting time*, in Proceedings of the 3rd Annual International Workshop on Approximation Algorithms for Combinatorial Optimization Problems (APPROX'00), 2000, pp. 226–237.

THE COMPUTATIONAL COMPLEXITY OF TUTTE INVARIANTS FOR PLANAR GRAPHS*

DIRK VERTIGAN†

Abstract. For each pair of algebraic numbers (x, y) , the complexity of computing the Tutte polynomial $T(G; x, y)$ of a planar graph G is determined. This computation is found to be $\#\overline{P}$ -complete except when $(x - 1)(y - 1) = 1, 2$ or when (x, y) is one of $(1, 1)$, $(-1, -1)$, (j, j^2) , or (j^2, j) , where $j = e^{2\pi i/3}$, in which case it is polynomial time computable. A corollary gives the computational complexity of various enumeration problems for planar graphs.

Key words. Tutte polynomial, knot polynomials, planar graphs, computational complexity, polynomial time, $\#\overline{P}$ -complete, enumeration, reliability, percolation, Potts model

AMS subject classifications. 05B35 (matroids), 68Q15 (complexity classes), 68R10 (graph theory), 82A43 (percolation)

DOI. 10.1137/S0097539704446797

1. Introduction. The main result of this paper is Theorem 5.1, which states the complexity of computing the Tutte polynomial $T(G; x, y)$ of a planar graph G at any fixed algebraic point or algebraic curve in the (x, y) -plane. It is $\#\overline{P}$ -complete (“computationally hard”) almost everywhere, that is, except when $(x - 1)(y - 1) = 1, 2$ or when (x, y) is one of $(1, 1)$, $(-1, -1)$, (j, j^2) , or (j^2, j) , where $j = e^{2\pi i/3}$, in which case it is polynomial time computable (“computationally easy”).

The corresponding result was done for the input class of general graphs in Proposition 1 of [8], where the complexity differed only on the curve $(x - 1)(y - 1) = 2$. Strengthening the result to the smaller input class of planar graphs requires that some novel, and more intricate, techniques be combined with those already used in [8].

One motivation for examining the planar graph case is that many interesting graph invariants are obtained as evaluations of the Tutte polynomial, giving the complexity of computing these invariants for planar graphs as an immediate corollary. For instance, for the input class of planar graphs, it is $\#\overline{P}$ -complete to count any one of the following: k -colorings, k -flows (for any fixed $k \in \{3, 4, \dots\}$), independent sets (forests), spanning sets, or acyclic orientations. And there is an equivalent complexity for one-variable polynomials such as the (all terminal) connectedness reliability polynomial, the percolation probability polynomial, the q -state Potts model polynomial (for any fixed $q \in \{3, 4, \dots\}$), as well as, of course, the full two-variable Tutte polynomial, and almost all specializations of these.

Another motivation is that the planar graph result is an important step toward answering the analogous question for the Homfly and Kauffman polynomials for knots and links; see [19].

The paper is arranged with the following sections, some of which have self-explanatory titles: 1. Introduction, 2. The Tutte polynomial, 3. Computational complexity, 4. Tutte invariants, 5. The main results, 6. Pointed Tutte polynomials, 7. The radial construction, 8. Some $4 \cdot k$ -tiles, 9. The radial construction using just T_{\setminus} and

*Received by the editors December 31, 2004; accepted for publication (in revised form) August 30, 2005; published electronically January 27, 2006.

<http://www.siam.org/journals/sicomp/35-3/44679.html>

†Department of Mathematics, Louisiana State University, Baton Rouge, LA 70803–4918 (vertigan@math.lsu.edu).

T_{\succ} , 10. Some useful combinatorial identities, 11. The main constructions, 12. The remainder of the proofs, and 13. Conclusions.

As a historical note, this work was done in 1989 and appeared in the author’s DPhil thesis [17].

2. The Tutte polynomial. For graph and matroid theory terminology and notation, we follow [21] or [12]. The Tutte polynomial is a two-variable polynomial invariant defined originally for graphs [15, 23] but later extended to matroids [2, 3].

A *matroid* M consists of a pair $M = (E, \rho)$ with finite *ground set* E and *rank function* $\rho : 2^E \rightarrow \mathbb{Z}$ satisfying certain axioms; see [21]. With any graph G we associate its *cycle matroid* $M(G) = (E(G), \rho_G)$ as follows.

Let $G = (V, E)$ be a graph. For $A \subseteq E$ let $V(A)$ be the set of vertices incident with edges in A . Let $G|A$ (respectively, $G||A$) be the subgraph of G consisting of the edges in A and the vertices in $V(A)$ (respectively, V). (Thus $G||A$ may contain isolated vertices which are deleted to obtain $G|A$.) Let $c(G)$ denote the number of components of G . The *rank*, $\rho(G)$, of G is given by

$$(2.1) \quad \rho(G) = |V(G)| - c(G).$$

The *cycle matroid*, $M(G)$, of G is defined to be $M(G) = (E(G), \rho_G)$, where for $A \subseteq E$,

$$(2.2) \quad \rho_G(A) = \rho(G|A) = |V(A)| - c(G|A) = \rho(G||A) = |V| - c(G||A).$$

If the graph is directed, then the associated matroid is independent of the orientation of edges. A matroid of the form $M(G)$ is a *graphic matroid*. Let \mathbb{G} and \mathbb{PG} denote the classes of (cycle matroids of) graphs and planar graphs, respectively.

The *Tutte polynomial* of a matroid $M = (E, \rho)$, with ground set E and rank function $\rho : 2^E \rightarrow \mathbb{Z}$, is defined to be

$$(2.3) \quad T(M; x, y) = \sum_{A \subseteq E} (x - 1)^{\rho(E) - \rho(A)} (y - 1)^{|A| - \rho(A)}.$$

The Tutte polynomial of a graph G is the same as that of the matroid $M(G)$. We sometimes abuse notation by writing G instead of $M(G)$.

A matroid has a unique partition into *connected components*, and for a graph G the components of $M(G)$ are the edge sets of blocks. In particular, G is 2-connected if and only if $M(G)$ is connected. If a matroid $M = (E, \rho)$ has connected components $(E_i : i \in I)$, then

$$(2.4) \quad T(M; x, y) = \prod_{i \in I} T(M|E_i; x, y);$$

see [21] for more information. In particular, for a graph, its Tutte polynomial is simply the product of the Tutte polynomial of its blocks. Because of this simple formula we can almost always restrict attention to 2-connected graphs.

3. Computational complexity. For terminology and notation in computational complexity, we follow [5, 16]; see also [6, 17, 22]. Every complexity class we consider will be a class of *functions* $f : \{0, 1\}^* \rightarrow \{0, 1\}^*$, where $\{0, 1\}^*$ is the set of binary strings. Denote the *size* (or *length*) of a string $w \in \{0, 1\}^*$ by $\|w\|$. We assume that the objects we consider, such as graphs, matrices, rationals, polynomials, etc., are all encoded as binary strings in a standard way.

Let FP denote the class of *polynomial time computable functions*. Thus $f \in \text{FP}$ means there is a deterministic algorithm such that for every $w \in \{0, 1\}^*$, the algorithm outputs $f(w)$ with the number of steps taken bounded by a polynomial in $\|w\|$.

For functions f, g , we let $f \leq_T g$ denote that the function f is *polynomial time Turing reducible to g* . That is, there is a deterministic *g -oracle* algorithm to compute f in polynomial time. A *g -oracle* algorithm is an algorithm which is allowed to use the function g at a cost of only one step per use. Loosely speaking, if $f \leq_T g$, then f being (computationally) hard implies g is hard, while g being (computationally) easy implies f is easy.

For a class of functions \mathcal{F} , we let $\text{FP}^{\mathcal{F}}$ denote $\{f : f \leq_T g \text{ for some } g \in \mathcal{F}\}$. Also g is \mathcal{F} -complete if $g \in \mathcal{F}$ and $\text{FP}^{\mathcal{F}} = \text{FP}^{\{g\}}$; that is, $f \leq_T g$ for every $f \in \mathcal{F}$.

The class $\#\text{P}$ was first defined in [16]. A function $g : A \times B \rightarrow \{0, 1\}$ is *polynomially balanced* if, for any $(a, b) \in A \times B$ such that $g(a, b) \neq 0$, the size of b is polynomially bounded in the size of a (in particular, for each $a \in A$, there are only finitely many such b). A function $f : A \rightarrow \mathbb{N}$ is in $\#\text{P}$ if there exists a function

$$(3.1) \quad g : A \times B \rightarrow \{0, 1\}$$

which is polynomially balanced and polynomial time computable, and is such that for any $a \in A$,

$$(3.2) \quad f(a) = |\{b : b \in B, g(a, b) = 1\}| = \sum_{b \in B} g(a, b);$$

that is, on input a , f counts the number of b 's such that $g(a, b) = 1$. For example, counting the number of circuits in a graph is in $\#\text{P}$ since, given a graph a and a subset of its edges b , it is easy to check in polynomial time whether the set of edges is a circuit.

The two main classes we consider are FP and $\#\text{P}$. Actually, since most evaluations of the Tutte polynomial are not literally counting problems, we often use the class $\text{FP}^{\#\text{P}}$, the class of functions polynomial time Turing reducible to a function in $\#\text{P}$. We abbreviate $\text{FP}^{\#\text{P}}$ to $\overline{\#\text{P}}$.

The relation \leq_T is a quasi order; that is, it is reflexive and transitive. Within $\overline{\#\text{P}}$, the \leq_T -minimal elements are the functions in FP , while the \leq_T -maximal elements are the $\overline{\#\text{P}}$ -complete functions. All the functions considered in this paper are shown either to be $\overline{\#\text{P}}$ -complete or to be in FP .

Note that $\overline{\#\text{P}}$ -complete functions are NP-hard but are in Pspace.

4. Tutte invariants. Let \mathcal{C} be a class of matroids. Assume that any matroid M in \mathcal{C} can be encoded as a string in such a way that the size of the encoding of $M = (E, \rho) \in \mathcal{C}$ is polynomially bounded in $|E|$ (so that the input size can be taken to be $|E|$) and that the rank of any $A \subseteq E$ can be determined in time polynomially bounded in $|E|$. This assumption applies to all the classes of matroids mentioned in this paper (but not to the class of all matroids since there are too many of any given size).

Define the function

$$(4.1) \quad \tau^2(\mathcal{C}) : \mathcal{C} \rightarrow \mathbb{Z}[x, y], \text{ where } M \mapsto T(M; x, y).$$

This function determines the Tutte polynomial of each matroid $M \in \mathcal{C}$ on the whole (x, y) -plane.

Let \mathbb{A} denote the set of algebraic numbers. For any algebraic point $(x_0, y_0) \in \mathbb{A}^2$ define the function

$$(4.2) \quad \tau^0(\mathcal{C}, x_0, y_0) : \mathcal{C} \rightarrow \mathbb{Q}(x_0, y_0), \text{ where } M \mapsto T(M; x_0, y_0).$$

This function determines the Tutte polynomial of each matroid $M \in \mathcal{C}$ at a single point (x_0, y_0) .

It is also of interest to consider the Tutte polynomial of a matroid “along” an algebraic curve in the (x, y) -plane. Here is a formal definition (which will soon be made less formal). Let $p \in \mathbb{Q}[x, y]$ be an irreducible polynomial, let $\langle p(x, y) \rangle$ be the ideal generated by $p(x, y)$, and let $K = \langle (x, y) : p(x, y) = 0 \rangle$ be the corresponding algebraic curve. Define the function

$$(4.3) \quad \tau^1(\mathcal{C}, K) : \mathcal{C} \rightarrow \mathbb{Q}[x, y]/\langle p(x, y) \rangle, \text{ where } M \mapsto T(M; x, y) + \langle p(x, y) \rangle.$$

This function gives $T(M; x, y)$ reduced modulo $p(x, y)$ (in some predetermined canonical form). If the curve is expressed as a rational curve $\langle (x(s), y(s)) : s \in \mathbb{A} \rangle$, where x and y are rational functions of s , we can alternatively state

$$(4.4) \quad \tau^1(\mathcal{C}, K) \text{ sends } M \in \mathcal{C} \text{ to } T(M; x(s), y(s)),$$

the latter being a rational function of s . Thus, strictly speaking, the definition of $\tau^1(\mathcal{C}, K)$ depends on the representation of the curve K . However, it is easily shown that one can translate between any two of these representations in polynomial time, so that the computational complexity of $\tau^1(\mathcal{C}, K)$ is independent of the representation of K .

The polynomial time bounds on the algorithms and reductions presented in this paper will be fairly evident to those readers familiar with computational complexity except possibly for algebraic manipulations with the outputs of the Tutte invariants. But note, for example, that the outputs of $\tau^0(\mathcal{C}, x_0, y_0)$ are not arbitrary algebraic numbers but are elements of $\mathbb{Q}(x_0, y_0)$, which is a finite degree extension of \mathbb{Q} and whose elements can be represented as rational vectors of a fixed finite dimension. More generally, outputs of $\tau^1(\mathcal{C}, K)$ and $\tau^2(\mathcal{C})$ can be expressed as rational vectors of a dimension that is polynomially bounded in input size. In all cases, the rationals themselves have a polynomially bounded encoding as strings. And finally, considering the types of issues addressed in [6], these bounds remain true with all the algebraic manipulations done, primarily solving linear equations, as is needed for Lagrange interpolation.

Observe that if (x_0, y_0) is an algebraic point and K is an algebraic curve containing (x_0, y_0) , then trivially,

$$(4.5) \quad \tau^0(\mathcal{C}, x_0, y_0) \leq_T \tau^1(\mathcal{C}, K) \leq_T \tau^2(\mathcal{C}).$$

In certain cases (see Theorems 4.2 and 4.3 below) these reducibilities can be reversed, motivating the following definitions and terminology.

DEFINITION 4.1. *For each $q \in \mathbb{A}$ define the curves $H_q = \langle (x, y) : (x - 1)(y - 1) = q \rangle$ and define $H_0^x = \langle (x, y) : x = 1 \rangle$ and $H_0^y = \langle (x, y) : y = 1 \rangle$. The curves H_0^x , H_0^y , and H_q for $q \in \mathbb{A} - \{0\}$ are called special curves. Let $j = e^{2\pi i/3}$. The points $(1, 1)$, $(0, 0)$, $(-1, -1)$, $(0, -1)$, $(-1, 0)$, $(i, -i)$, $(-i, i)$, (j, j^2) , (j^2, j) are called special points.*

The following results come from [8]. It is assumed that the class of matroids \mathcal{C} satisfies certain natural technical conditions given in [8] and addressed in detail in

[17]. All the classes of matroids in this paper easily satisfy these conditions, allowing the use of Theorems 4.2 and 4.3 below.

The following is Theorem 1 from [8]. (This was proved only when the curve is expressed as a rational curve, but it is straightforward to extend this, though this extension is not used here.)

THEOREM 4.2. *If K is an algebraic curve, then $\tau^2(\mathcal{C}) \leq_T \tau^1(\mathcal{C}, K)$ unless K is a special curve.*

The following is Theorem 2 from [8].

THEOREM 4.3. *If K is a special curve and $(x, y) \in K$, then $\tau^1(\mathcal{C}, K) \leq_T \tau^0(\mathcal{C}, x, y)$ unless (x, y) is a special point.*

The following is Proposition 1 of [8]. The above two theorems are very useful for proving this proposition, as well as the main Theorem 5.1. But while Proposition 4.4 and Theorem 5.1 look quite similar—in changing the input class from \mathbb{G} to $\mathbb{P}\mathbb{G}$, the computational complexity changes only on the special curve H_2 —the proofs have significant differences, and Theorem 5.1 makes essentially no use of Proposition 4.4.

PROPOSITION 4.4.

- (i) *The function $\tau^2(\mathbb{G})$ is $\overline{\#\mathbb{P}}$ -complete.*
- (ii) *If K is an algebraic curve, then $\tau^1(\mathbb{G}, K)$ is $\overline{\#\mathbb{P}}$ -complete unless $K = H_1$, in which case $\tau^1(\mathbb{G}, K)$ is in FP.*
- (iii) *If $(x, y) \in \mathbb{A}^2$ is an algebraic point, then $\tau^0(\mathbb{G}, x, y)$ is $\overline{\#\mathbb{P}}$ -complete unless $(x, y) \in H_1$ or (x, y) is a special point, in which case $\tau^0(\mathbb{G}, x, y)$ is in FP.*

Note that for a graph G the Ising partition function of G (see [4, 11, 9, 22]) is given (up to an easily computable factor) by the Tutte polynomial of $M(G)$ along the curve H_2 (see, for example, [22]). By [9], this is $\overline{\#\mathbb{P}}$ -complete for graphs, whereas by [4, 11], this is polynomial time computable for planar graphs.

5. The main results. The main computational complexity result, proved in section 12, is the following.

THEOREM 5.1.

- (i) *The function $\tau^2(\mathbb{P}\mathbb{G})$ is $\overline{\#\mathbb{P}}$ -complete.*
- (ii) *If K is an algebraic curve, then $\tau^1(\mathbb{P}\mathbb{G}, K)$ is $\overline{\#\mathbb{P}}$ -complete unless $K = H_1$ or $K = H_2$, in which case $\tau^1(\mathbb{P}\mathbb{G}, K)$ is in FP.*
- (iii) *If $(x, y) \in \mathbb{A}^2$ is an algebraic point, then $\tau^0(\mathbb{P}\mathbb{G}, x, y)$ is $\overline{\#\mathbb{P}}$ -complete unless $(x, y) \in H_1 \cup H_2$ or (x, y) is a special point, in which case $\tau^0(\mathbb{P}\mathbb{G}, x, y)$ is in FP.*

Observe that the only special points not on H_1 or H_2 are $(1, 1)$, $(-1, -1)$, (j, j^2) , and (j^2, j) .

The paper [8] and the book [22] list many well-known quantities associated with graphs which are obtained (up to an easily computable factor) by evaluating the Tutte polynomial at certain points and curves. The above theorem shows that most of these calculations are $\overline{\#\mathbb{P}}$ -complete even for planar graphs. Some examples are given in the following corollary.

COROLLARY 5.2. *The following evaluations are $\overline{\#\mathbb{P}}$ -complete.*

$\tau^0(\mathbb{P}\mathbb{G}, 1 - k, 0)$: counting the number of k -colorings of a planar graph for any fixed $k \in \{3, 4, \dots\}$. Note that, by the four-color theorem, the corresponding decision problem is trivial for $k \geq 4$.

$\tau^0(\mathbb{P}\mathbb{G}, 0, 1 - k)$: counting the number of k -flows of a planar graph for any fixed $k \in \{3, 4, \dots\}$. As above, the corresponding decision problem is trivial for $k \geq 4$.

$\tau^0(\mathbb{P}\mathbb{G}, 2, 1)$ or $\tau^0(\mathbb{P}\mathbb{G}, 1, 2)$: counting the number of independent sets (forests) or spanning sets in a planar graph (shown independently by Jerrum [10]).

$\tau^1(\mathbb{P}\mathbb{G}, H_0^x)$ or $\tau^0(\mathbb{P}\mathbb{G}, 1, (1 - p)^{-1})$: “(all terminal) connectedness reliability” or “percolation probability” for planar graphs for any fixed $p \in \mathbb{A} - \{0, 1\}$. (Compare this to the “two-terminal connectedness reliability,” which is given (up to an easily computable factor) by the *pointed* Tutte polynomial T_L along the curve H_1 (see section 6 and (6.9)). This was shown to be $\overline{\#P}$ -complete in [13].)

$\tau^0(\mathbb{P}\mathbb{G}, 2, 0)$: counting the number of acyclic orientations of a planar graph.

$\tau^1(\mathbb{P}\mathbb{G}, H_q)$ or $\tau^0(\mathbb{P}\mathbb{G}, 1 + \frac{q}{x-1}, x)$: the “ q -state Potts model” for planar graphs, for any fixed $q \in \{3, 4, \dots\}$, $x \in \mathbb{A} - \{1\}$, $(q, x) \notin \{(3, j), (3, j^2), (4, -1)\}$ ($j = e^{2\pi i/3}$).

Moreover, Theorem 5.1 yields some immediate corollaries about the computational complexity of certain knot and link invariants (using connections in [7, 14]), while more comprehensive results are obtained with extra work which nevertheless has Theorem 5.1 as an essential foundation; see [19].

6. Pointed Tutte polynomials. The pointed Tutte polynomials [1] have important uses in [8, 18, 17]. In those works they appear in conjunction with the *matroid tensor product* operation [1], which we do not define here, although we note that it is crucial for Theorems 4.2 and 4.3, which we certainly use.

We primarily need pointed Tutte polynomials, and the sets defined in (6.1) below, in various places such as Lemmas 9.1, 11.2, 12.1, and 12.6. They are needed to fill what would otherwise be a “gap” as discussed preceding Lemma 11.2.

For $M = (E, \rho)$ and $g \in E$ define

$$(6.1) \quad \begin{aligned} C(M, g) &= \{A : g \in A, A \subseteq E, \rho(A) = \rho(A - \{g\}) + 1\}, \\ L(M, g) &= \{A : A \subseteq E - \{g\}, \rho(A \cup \{g\}) = \rho(A)\}. \end{aligned}$$

The *pointed* Tutte polynomials [1] are defined by

$$(6.2) \quad T_C(M, g; x, y) = \sum_{A \in C(M, g)} (x - 1)^{\rho(E) - \rho(A)} (y - 1)^{|A| - \rho(A)},$$

$$(6.3) \quad T_L(M, g; x, y) = \sum_{A \in L(M, g)} (x - 1)^{\rho(E) - \rho(A)} (y - 1)^{|A| - \rho(A)}.$$

Let \mathcal{C} be a class of matroids, and let $\overline{\mathcal{C}} = \{(M, g) : M = (E, \rho) \in \mathcal{C}, g \in E\}$. Let (x, y) be an algebraic point and let K be an algebraic curve. Define the function

$$(6.4) \quad \tau_L^2(\mathcal{C}) : \overline{\mathcal{C}} \rightarrow \mathbb{Z}[x, y], \text{ where } (M, g) \mapsto T_L(M, g; x, y),$$

and define the functions $\tau_L^1(\mathcal{C}, K)$, $\tau_L^0(\mathcal{C}, x, y)$, $\tau_C^2(\mathcal{C})$, $\tau_C^1(\mathcal{C}, K)$, $\tau_C^0(\mathcal{C}, x, y)$ in the obvious way (analogous to (4.1)–(4.4)).

When x, y, M, g are known from the context, it is convenient to abbreviate $T(M; x, y)$, $T(M \setminus g; x, y)$, $T(M/g; x, y)$, $T_C(M, g; x, y)$, and $T_L(M, g; x, y)$ to T, T', T'', T_C , and T_L , respectively. These are related by various identities [1] (or [17]). For instance, if g is not a loop or coloop, then

$$(6.5) \quad \begin{pmatrix} T' \\ T'' \end{pmatrix} = \begin{pmatrix} (x - 1) & 1 \\ 1 & (y - 1) \end{pmatrix} \begin{pmatrix} T_C \\ T_L \end{pmatrix}.$$

Note that the 2×2 matrix in (6.5) is nonsingular if and only if $(x, y) \neq 1$, that is, $(x, y) \notin H_1$. Using this and sundry other facts, it is easily shown that if $(x, y) \notin H_1$ and

\mathcal{C} is closed under *minors* (see [21]), then $\tau_L^0(\mathcal{C}, x, y)$ and $\tau_C^0(\mathcal{C}, x, y)$ are computationally equivalent to $\tau^0(\mathcal{C}, x, y)$, and similarly for τ^1 and τ^2 .

However, if $(x, y) \in H_1$, then $\tau^0(\mathcal{C}, x, y)$ is in FP since for $M = (E, \rho) \in \mathcal{C}$, $T(M; x, y)$ is simply $(x - 1)^{\rho(E)}y^{|E|}$. But, for example, if $(x, y) \in H_1 - \{(0, 0)\}$, then $\tau_L^0(\mathbb{P}\mathbb{G}, x, y)$ and $\tau_C^0(\mathbb{P}\mathbb{G}, x, y)$ are $\overline{\#P}$ -complete by the following reasoning. It is clear from (6.3) that

$$(6.6) \quad T_L(M, g; 2, 2) = |L(M, g)|.$$

Consider a graph $G = (V, E)$, edge $g \in E$, and subset $A \subseteq E - \{g\}$. Now $A \in L(M(G), g)$ if and only if $G|A$ contains a path between the ends of g , by the equivalence (1) \iff (4) in Lemma 9.1. Provan [13] shows that, given a planar graph G and edge g , determining the number of such subsets is $\overline{\#P}$ -complete. Since this number is $T_L(M(G), g; 2, 2)$, it follows that

$$(6.7) \quad \tau_L^0(\mathbb{P}\mathbb{G}, 2, 2) \text{ is } \overline{\#P}\text{-complete.}$$

Also $\tau_C^0(\mathbb{P}\mathbb{G}, 2, 2)$ is $\overline{\#P}$ -complete since $T_C(M(G), g; 2, 2) = 2^{|E|-1} - T_L(M(G), g; 2, 2)$. Using an argument similar to that in the proof of Theorem 4.3, it follows that $\tau_L^0(\mathbb{P}\mathbb{G}, x, y)$ and $\tau_C^0(\mathbb{P}\mathbb{G}, x, y)$ are $\overline{\#P}$ -complete whenever $(x, y) \in H_1 - \{(0, 0)\}$.

The following polynomial, related to the *2-terminal connectedness reliability*, is defined in [13] (where s and t are the ends of edge g):

$$(6.8) \quad R(G, s, t; p) = \sum_{A \in L(M(G), g)} p^{|A|}(1 - p)^{|E-A|}.$$

Using (6.3), it follows that

$$(6.9) \quad R(G, s, t; p) = p^{\rho(E)}(1 - p)^{|E|-\rho(E)}T_L(M(G), g; p^{-1}, (1 - p)^{-1}).$$

Observe that $(p^{-1}, (1 - p)^{-1}) \in H_1$ for all $p \in \mathbb{A} - \{0, 1\}$. Thus finding 2-terminal connectedness reliability is $\overline{\#P}$ -complete even for any fixed $p \in \mathbb{A} - \{0, 1\}$.

Finally note that since $\tau_L^2(\mathbb{P}\mathbb{G}) \leq_T \tau^2(\mathbb{P}\mathbb{G})$ (see comments following (6.5)) and $\tau_L^0(\mathbb{P}\mathbb{G}, 2, 2) \leq_T \tau_L^2(\mathbb{P}\mathbb{G})$ (similarly to (4.5)), it follows that

$$(6.10) \quad \tau_L^0(\mathbb{P}\mathbb{G}, 2, 2) \leq_T \tau^2(\mathbb{P}\mathbb{G}).$$

7. The radial construction. In what follows we regard plane graphs as being drawn on the 2-sphere. We describe a construction to piece together pieces of plane graphs. This construction is also important in [19].

Let G be a 2-connected directed plane graph, with vertex, edge, and face sets $V = V(G)$, $E = E(G)$, and $F = F(G)$. The *radial graph* of G , denoted G° , is a simple bipartite plane graph constructed as follows. It is instructive to regard the two graphs as being drawn simultaneously on the same 2-sphere. The vertex set of G° is $V \cup F$, with each vertex $f \in F$ of G° drawn in the face $f \in F$ of G . There is an edge, drawn in the obvious way, of G° between vertices $v \in V$ and $f \in F$ of G° if and only if vertex $v \in V$ is incident with face $f \in F$ in graph G . Note that every face of G° is a square, that is, it is bounded by four edges (regardless of actual shape in a drawing), and we identify the face set $F(G^\circ)$ of G° with the edge set $E = E(G)$ of G .

Figure 7.1 depicts a particular four-edge graph G (thick directed edges) and its radial graph G° (thin edges) with its corresponding four square faces. The three

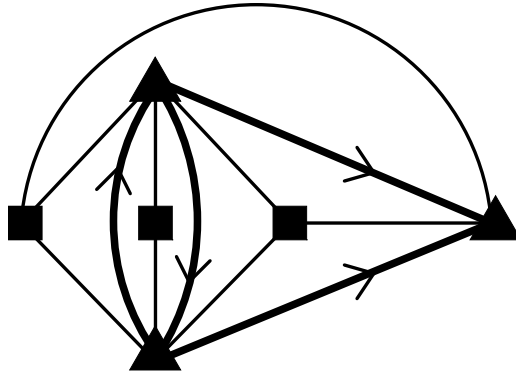


FIG. 7.1. Graph G (thick edges) and radial graph G^\diamond (thin edges).

vertices and three faces of G correspond to the six vertices of G^\diamond , which are depicted, respectively, as triangular and square dots.

We use the orientation of G to add some extra labelling. Define functions $h, t : E \rightarrow V$, where $h(e)$ and $t(e)$ are, respectively, the *head* and *tail* of the directed edge $e \in E$. Also define functions $r, l : E \rightarrow F$, where $r(e)$ and $l(e)$ are, respectively, the *right(face)* and *left(face)* of the directed edge $e \in E$, when viewing e as pointing *up* (toward the head). Thus, the vertices of G^\diamond incident with face $e \in F(G^\diamond)$ are $h(e)$, $r(e)$, $t(e)$, $l(e)$ in clockwise order.

When G and G^\diamond are drawn simultaneously on the same 2-sphere, then edge $e \in E$ of G appears as the diagonal of face $e \in F(G^\diamond)$, which connects $h(e)$ and $t(e)$ (see Figure 7.1 again). Observe that we can also draw simultaneously the *plane dual* G^* of G with vertex, edge, and face sets $V(G^*) = F(G)$, $E(G^*) = E(G)$, $F(G^*) = V(G)$, and with each edge $e \in E(G^*) = E(G)$ appearing as the diagonal of face $e \in F(G^\diamond)$, which connects $l(e)$ and $r(e)$.

A *disk graph* H with *boundary vertex set* $B \subseteq V(H)$ is a plane graph drawn on a (closed) disk D , such that the drawing intersects the boundary of D precisely in vertex set B . The vertices in $V(H) - B$ are called the *internal* vertices. We wish to glue disk graphs together into larger disk graphs or plane graphs; for this purpose the labelling of the boundary vertices is important, and we do not equate a disk graph with its rotations or reflections.

For a positive integer k , a $4 \cdot k$ -*tile* is a disk graph with B, D as above, together with a distinguished set of four *boundary points* P (that is $|P| = 4$) on the boundary of D , with $P \cap B = \emptyset$, where there are exactly $|B| = 4k$ boundary vertices in four groups of k separated by the four points in P . We will typically label the four boundary points h, r, t, l in clockwise order, or use related names. We treat the disk as a square with corner points h, r, t, l and call the sides hr, rt, tl, lh in the natural way. The $4k$ boundary vertices are labelled hr^i, rt^i, tl^i, lh^i for $i = 1, \dots, k$, and they appear with the four boundary points h, r, t, l in clockwise order as follows:

$$(7.1) \quad h, hr^1, \dots, hr^k, r, rt^k, \dots, rt^1, t, tl^1, \dots, tl^k, l, lh^k, \dots, lh^1.$$

For each $i = 1, \dots, k$, the boundary vertices hr^i, rt^i, tl^i, lh^i are said to be of *type* i . Observe that boundary vertices of type 1 are consecutive with boundary points h and t , while boundary vertices of type k are consecutive with boundary points r and l .

Figure 7.2 depicts the boundary labelling in the $k = 3$ case. Boundary vertices are

depicted as round dots, while boundary points are depicted as triangular dots (h and t) and square dots (r and l). Figures 8.1, 8.2, and 11.1(a), with $k = 3, 3, 2$, respectively, depict $4 \cdot k$ -tiles with boundary labels and boundary “diamond” omitted from the drawing, with the understanding that these omitted features are to be deduced in the obvious way.

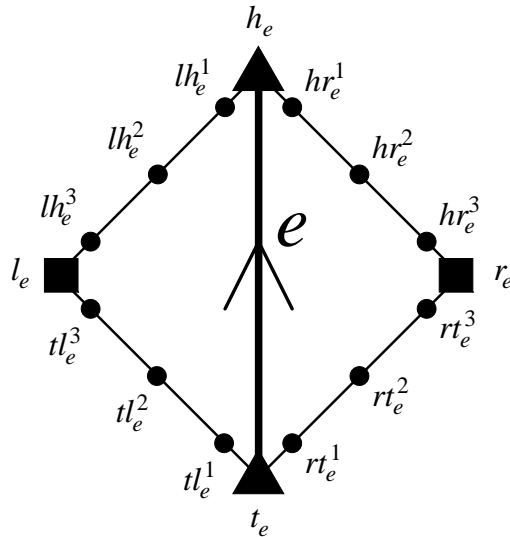


FIG. 7.2. Boundary labelling for a $4 \cdot k$ -tile ($k = 3$).

In the *radial construction* we place a $4 \cdot k$ -tile in each face of a radial graph and join them as we now detail. For visualization one may refer to Figures 7.1 and 7.2. Let G be a plane graph. Let k be a positive integer. For each $e \in E(G) = F(G^\circ)$ let T_e be a $4 \cdot k$ -tile drawn on a disk D_e with boundary points h_e, r_e, t_e, l_e in clockwise order. Other features of the $4 \cdot k$ -tile T_e also have an “ e ” incorporated into the notation where necessary.

DEFINITION 7.1. Given plane graph G , k , and $(T_e : e \in E)$ as above, where $E = E(G) = F(G^\circ)$, the radial construction produces a plane graph denoted

$$(7.2) \quad \Theta_G(T_e : e \in E)$$

by gluing the graphs T_e as follows. For each $e \in E$, place tile T_e in face $e \in F(G^\circ)$ of G° so that the boundary of disk D_e coincides with the boundary of face $e \in F(G^\circ)$. The boundary points h_e, r_e, t_e, l_e of T_e are identified with vertices $h(e), r(e), t(e), l(e)$ of G° . For each edge of G° incident with faces $e, g \in F(G^\circ)$, the appropriate k vertices of T_e are identified with the appropriate k vertices of T_g in the natural way.

Observe that only boundary vertices of the same *type* are identified.

We emphasize that $\Theta_G(T_e : e \in E)$ consists only of graphs $(T_e : e \in E)$ joined together as specified and does not include G° , or G or G^* , but instead these graphs determine *how* the T_e ’s are joined together. Nevertheless, it is instructive to visualize all four graphs as superimposed on the same 2-sphere.

This construction was called the *medial* construction in [17], but it is exactly the same construction. (The medial graph is the plane dual of the radial graph.) We remark that the vertices that are called $hr_e^i, rt_e^i, tl_e^i, lh_e^i$ in this paper are denoted in [17, 19] as $(e, h, a, i), (e, t, c, i), (e, t, a, i), (e, h, c, i)$, respectively.

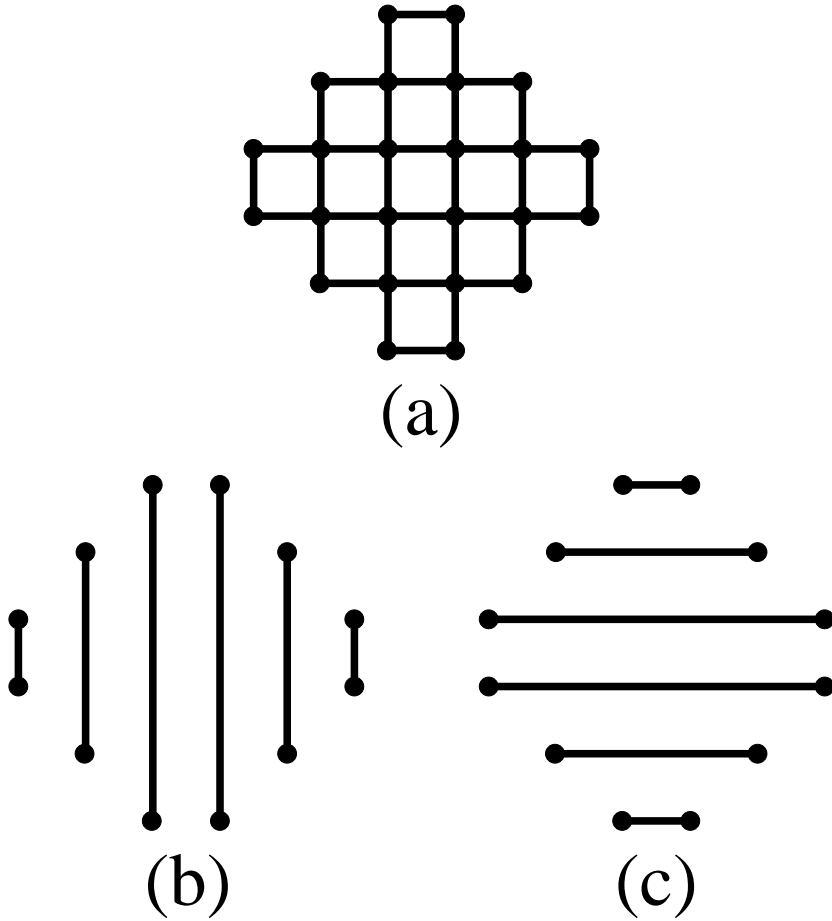


FIG. 8.1. (a) $T_{\#}^k$, (b) $T_{(c)}^k$, (c) T_{\geq}^k ($k = 3$).

8. Some $4 \cdot k$ -tiles. We now describe some $4 \cdot k$ -tiles, which we shall need in various constructions and arguments. For convenience we initially define some as being drawn in the coordinate (X, Y) -plane with the X - and Y -axes being, respectively, horizontal and vertical. Let k be a positive integer. Let the disk D^k be the \diamond -shaped region of the (X, Y) -plane:

$$(8.1) \quad D^k = \{(X, Y) : |X| + |Y| \leq 2k\}.$$

The four distinguished boundary points of D^k are called h, r, t, l and have (X, Y) -coordinates $(0, 2k), (2k, 0), (0, -2k), (-2k, 0)$, respectively.

Let G_{\square}^{∞} be the infinite square grid in the (X, Y) -plane such that the vertices have coordinates (X, Y) , where X and Y are odd integers. The edges are parallel to the axes and have length 2. Call the edges *horizontal* and *vertical* in the obvious way.

Let $T_{\#}^k$ be the $4 \cdot k$ -tile on disk D^k with boundary points as above, where $T_{\#}^k$ is the intersection of G_{\square}^{∞} with D^k . See Figure 8.1(a) for the $k = 3$ case. The $4k$ boundary vertices are those which lie on the boundary of D^k , that is, the vertices with coordinates (X, Y) , where X and Y are odd integers and $|X| + |Y| = 2k$. For example, vertex hr^i has coordinates $(2i - 1, 2k - 2i + 1)$ for $i = 1, \dots, k$.

Observe that $T_{\#}^k$ has $4k^2$ edges and $2k(k+1)$ vertices, namely, $4k$ degree-2 boundary vertices and $2k(k-1)$ degree-4 internal vertices.

DEFINITION 8.1. Let T_{\langle}^k be the $4 \cdot k$ -tile obtained from $T_{\#}^k$ by restricting to the vertical edges. That is, T_{\langle}^k consists purely of vertex disjoint paths connecting boundary vertices hr^i to rt^i and tl^i to lh^i for $i = 1, \dots, k$.

Let T_{\geq}^k be the $4 \cdot k$ -tile obtained from $T_{\#}^k$ by restricting to the horizontal edges. That is, T_{\geq}^k consists purely of vertex disjoint paths connecting boundary vertices hr^i to lh^i and tl^i to rt^i for $i = 1, \dots, k$.

For our purposes the path lengths will generally not matter and we may use the notations T_{\langle}^k and T_{\geq}^k as long as boundary vertices are joined pairwise as above, regardless of (nonzero) path lengths. Figure 8.1(b)–(c) depicts T_{\langle}^k and T_{\geq}^k with $k = 3$ and with (internal) degree-2 vertices suppressed.

For a graph G , a *decontraction* at vertex v by new edge e yields a graph G' such that $G'/e = G$ and edge e in G' has endpoints, say v_1 and v_2 , in G' that are shrunk to v when contracting e . To uniquely specify G' we need to specify G , v , e together with a partition of the edges incident with v into two parts, specifying which of these edges are to be incident with v_1 or v_2 in G' . A degree d vertex can be decontracted into vertices of degree d_1 and d_2 provided $d = d_1 + d_2 - 2$. If G is drawn on a surface, then we can locally modify this drawing to obtain a drawing of G' on the surface provided the abovementioned partition is appropriately compatible with the cyclic ordering of edges around v . Thus there are three ways to decontract a degree-4 vertex into two degree-3 vertices, but only two are compatible with a surface drawing.

DEFINITION 8.2. We obtain a $4 \cdot k$ -tile $\tilde{T}_{\#}^k$ by decontracting at all of the $2k(k+1)$ vertices of $T_{\#}^k$, as follows. For future reference we will color the edges of $T_{\#}^k$ blue, while the new edges in $\tilde{T}_{\#}^k$ arising from decontraction are colored red. We require that the following hold.

- (a) Each of the $4k$ degree-2 boundary vertices of $T_{\#}^k$ is decontracted into a degree-1 boundary vertex and a degree-3 internal vertex in a unique way.
- (b) Each of the $2k(k-1)$ degree-4 internal vertices of $T_{\#}^k$ is decontracted into two degree-3 internal vertices in one of the two ways compatible with the plane drawing.
- (c) The decontractions in (b) are chosen such that the $4k^2$ blue edges form a single circuit.

Readers can verify that (c) can be satisfied in many ways, for instance, by doing one way of decontracting at the degree-4 internal vertices with coordinates (X, Y) satisfying $Y - X = 2k - 2$ (so that the corresponding red edges are roughly parallel to sides hr and tl), and the other way of decontracting at the rest (so that the corresponding red edges are roughly parallel to sides rt and lh). See Figure 8.2 for $\tilde{T}_{\#}^k$ with $k = 3$, and with this choice of decontraction. A fully explicit description of $\tilde{T}_{\#}^k$ (under a different name) is given in [17]. For the purposes of this paper, any choice in (b) that satisfies (c) will work for the constructions in which it appears. The blue edges retain the “vertical” or “horizontal” labelling they inherit from $T_{\#}^k$. (The red edges are just red.)

The (X, Y) -coordinates were convenient for the above definitions but we now no longer need them. We now ignore these coordinates and treat our graphs and tiles topologically or combinatorially.

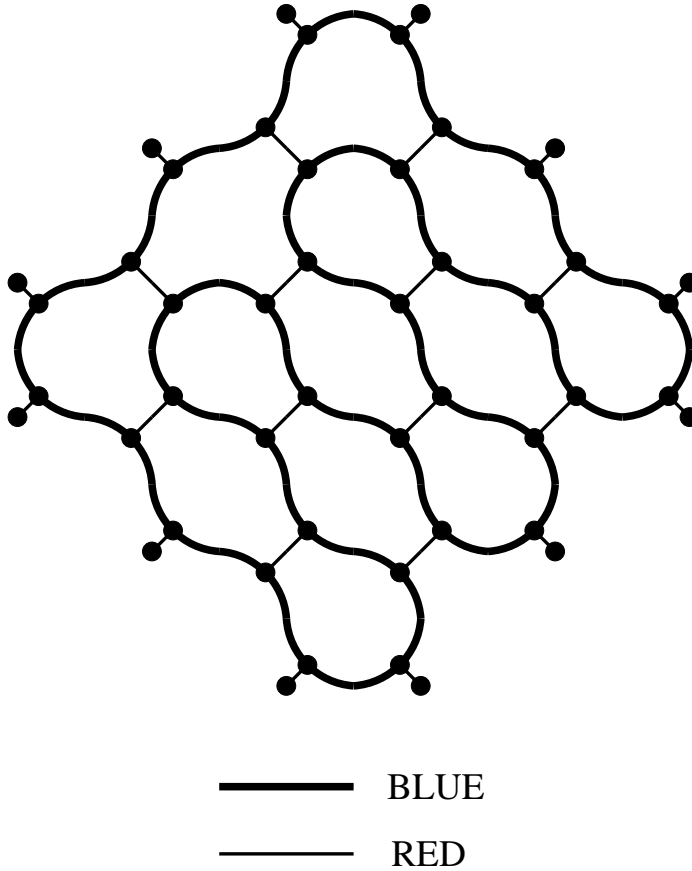


FIG. 8.2. $\tilde{T}_{\#}^k$ with $k = 3$, and with a valid choice of decontractions as described in the text. Thick edges are blue. Thin edges are red. See Definition 8.2.

9. The radial construction using just T_{\langle}^k and T_{\succ}^k . We first consider the radial construction when all tiles are of the form T_{\langle}^k and T_{\succ}^k . Let G be a directed plane graph, as above. For $A \subseteq E$ and a positive integer k , construct the plane graph G_A^k as follows. Suppose

$$(9.1) \quad G_A^k = \Theta_G(T_e : e \in E), \text{ where } T_e = \begin{cases} T_{\langle}^k & \text{for } e \in A, \\ T_{\succ}^k & \text{for } e \in E - A. \end{cases}$$

Actually this means each T_e is a copy of T_{\langle}^k or T_{\succ}^k .

Let E_A^k be the edge set of G_A^k , and for $g \in E$ let $E_{A,g}^k$ be the set of edges in T_g .

Clearly G_A^k consists of some number, say $m_G(A, k)$ or $m(A, k)$, of vertex disjoint circuits. Moreover, for any $g \in E$, the boundary vertices of $4 \cdot k$ -tile T_g will be pairwise joined by vertex disjoint paths outside its disk in a way that can be specified depending on A . The next lemma provides more detail of this. This lemma highlights one crucial idea toward the main theorem, as well as in the connection between the Tutte polynomial for planar graphs and various invariants of link diagrams [19]; see also [7, 14].

Let $c(A)$ denote $c(G||A)$, which need not equal $c(G|A)$. For statements (4), (5), (6), (4'), (5'), (6') in Lemma 9.1 below, it does not matter whether we use “|” or “||”.

LEMMA 9.1. *Let G be a 2-connected plane graph, and let $M(G) = (E, \rho)$ be its cycle matroid. Let k be a positive integer. Let $A \subseteq E$. Let*

$$(9.2) \quad G_A^k = \Theta_G(T_e : e \in E), \text{ where } T_e = \begin{cases} T_{\cup}^k & \text{for } e \in A, \\ T_{\cap}^k & \text{for } e \in E - A. \end{cases}$$

Then G_A^k consists of $m_G(A, k) = m(A, k)$ vertex disjoint circuits, where

$$(9.3) \quad m_G(A, k) = m(A, k) = k(|V| - 2\rho(A) + |A|).$$

Moreover, if $g \in E$ and $A \subseteq E - \{g\}$, then the following conditions (1)–(7) are equivalent:

- (1) $A \in L(M(G), g)$.
- (2) $\rho(A \cup \{g\}) = \rho(A)$.
- (3) $c(A \cup \{g\}) = c(A)$.
- (4) *The ends of g are joined by a path in $G|A$.*
- (5) *The ends of g are not joined by a path in $G^*|A$.*
- (6) *In $G_A^k|(E_A^k - E_{A,g}^k)$ there are vertex disjoint paths between hr_g^i and rt_g^i and between tl_g^i and lh_g^i for $i = 1, \dots, k$.*
- (7) $m(A \cup \{g\}, k) - m(A, k) = k$.

If $g \in E$ and $A \subseteq E - \{g\}$, then the following conditions (1')–(7') are equivalent and are the negation of (1)–(7):

- (1') $A \in 2^{E-\{g\}} - L(M(G), g) = \{A \in 2^{E-\{g\}} : A \cup \{g\} \in C(M(G), g)\}$.
- (2') $\rho(A \cup \{g\}) = \rho(A) + 1$.
- (3') $c(A \cup \{g\}) = c(A) - 1$.
- (4') *The ends of g are not joined by a path in $G|A$.*
- (5') *The ends of g are joined by a path in $G^*|A$.*
- (6') *In $G_A^k|(E_A^k - E_{A,g}^k)$ there are vertex disjoint paths between hr_g^i and lh_g^i and between tl_g^i and rt_g^i for $i = 1, \dots, k$.*
- (7') $m(A \cup \{g\}, k) - m(A, k) = -k$.

Proof. As noted in section 7, only vertices of the same type are joined, and thus G_A^k consists of k disjoint subgraphs (one corresponding to each $i = 1, \dots, k$), each isomorphic to G_A^1 . Therefore $m(A, k) = k \times m(A, 1)$, and it suffices to prove the case $k = 1$.

Clearly $m(\emptyset, 1) = |V|$, since $\Theta_G(T_{\cap})$ consists of one circuit per vertex of G . Then (9.3) will follow by induction on $|A|$ once we have shown that (2) \iff (7) and (2') \iff (7').

Now (1) \iff (2) by (6.1) and (2) \iff (3) by (2.2). Clearly (3) \iff (4), since they hold exactly when the ends of g are in the same connected component of $G|A$. Similarly (1') \iff (2') \iff (3') \iff (4'). We show (3) \iff (5) and (3') \iff (5') by negating, dualizing, and changing A to $E - A - g$ in (3') \iff (4') and (3) \iff (4), respectively. Clearly (1') is the negation of (1).

Now (6) and (6') give the only ways that the boundary vertices of T_g can be joined by paths outside the disk for T_g , given that we are using only tiles T_{\cup}^k and T_{\cap}^k . If we regard G_A^k, G°, G , and G^* as being drawn simultaneously on the same 2-sphere, then by drawing edges $e \in E$ of G and G^* as “diagonals” of squares as discussed in section 7, we may easily arrange that edges $e \in A$ of G and edges $e \in E - A$ of G^* miss all the edges of G_A^k . Since exactly one of (4) or (5') must hold, then to miss these paths, this forces the connections as in (6) or (6'), respectively. Thus (4) \iff (6) and (5') \iff (6').

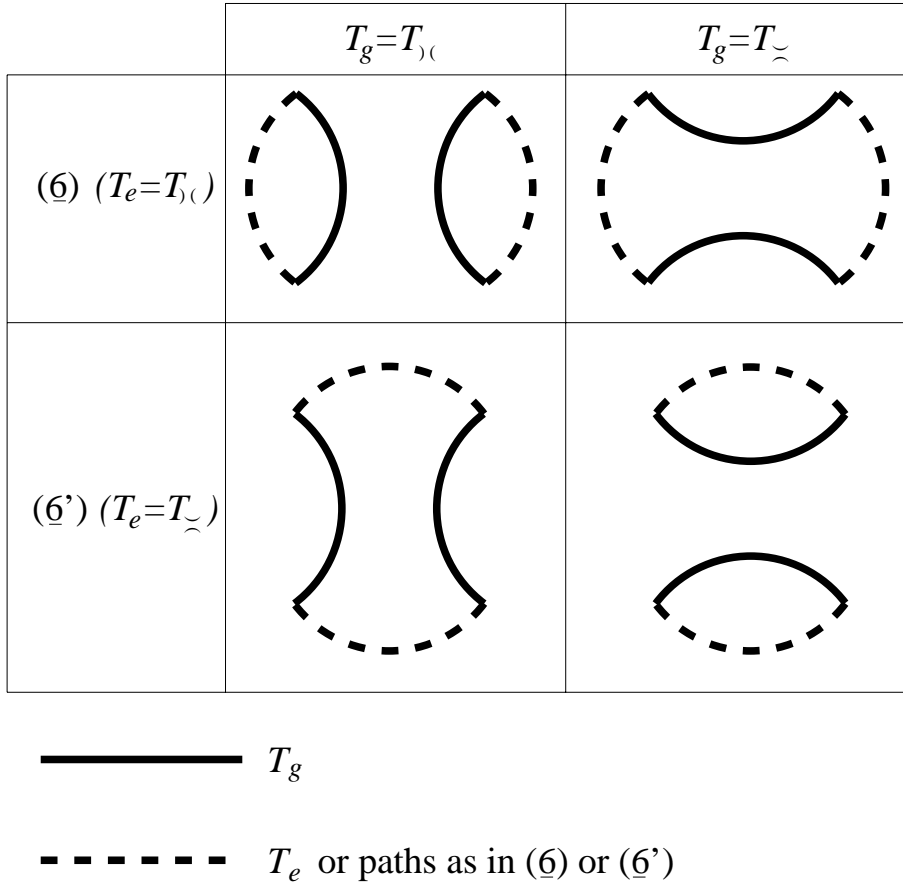


FIG. 9.1. How path routing relates to number of circuits.

To understand the remaining equivalences, with the help of Figure 9.1, first consider the special case where graph $G = G_{U12}$ consists of two parallel edges $E = \{e, g\}$ and $k = 1$. In that case, we can easily verify that (9.3) holds, that is,

$$(9.4) \quad m_{G_{U12}}(A, 1) = \begin{cases} 2 & \text{for } |A| = 0, 2, \\ 1 & \text{for } |A| = 1, \end{cases}$$

by considering all the appropriate ways of joining T_e and T_g , where each is either T_{\lrcorner} or T_{\sphericalangle} . Since, for the special case for $G = G_{U12}$, $e \in A \iff (1) - (6)$, we easily check that (6) \iff (7) and (6') \iff (7'). These equivalences follow for general G , since we need only consider path routings exactly as for the special case $G = G_{U12}$; that is, we need only consider how the boundary vertices of T_g are pairwise connected by vertex disjoint paths *internally* depending on whether T_g is T_{\lrcorner} or T_{\sphericalangle} , and how they are pairwise connected *externally* depending on whether (6) or (6') holds.

Figure 9.1 depicts the simple concept relating path routing to number of circuits. The same picture works for $G = G_{U12}$ (where dashed curves depict T_e being T_{\lrcorner} or T_{\sphericalangle}) and for general G (where dashed curves depict the paths referred to in (6) or (6')).

The general k case merely introduces a factor of k as noted above, and the proof is complete. \square

Remark 1. The above results hold regardless of the path lengths appearing in the T_{\subset} and T_{\supset} . Conditions (1)–(6) and (1′)–(6′) apply just as well when T_g is allowed to be an arbitrary $4 \cdot k$ -tile. The above results can also be restated for mathematical objects represented by curves in the plane such as *tangles* and *links* of topology; see [19].

10. Some useful combinatorial identities. We consider some useful combinatorial identities.

Lemma 10.1 below is central in the proof of Theorem 5.1. Let $M = (E, \rho)$ and $A \subseteq E$. The matroid M restricted to A , denoted $M|A$, is defined by $M|A = (A, \rho_A)$, where $\rho_A(B) = \rho(B)$ for all $B \subseteq A$.

LEMMA 10.1. *The following identity holds:*

$$(10.1) \quad \sum_{A \subseteq E} z^{|A|} (x-1)^{\rho(E)-\rho(A)} T(M|A; x, y) \\ = (1+z)^{|E|-\rho(E)} z^{\rho(E)} T\left(M; 1+(x-1)\frac{1+z}{z}, 1+(y-1)\frac{z}{1+z}\right).$$

Proof. By (2.3) the left-hand side of (10.1) is

$$(10.2) \quad \sum_{A \subseteq E} z^{|A|} (x-1)^{\rho(E)-\rho(A)} \sum_{B \subseteq A} (x-1)^{\rho(A)-\rho(B)} (y-1)^{|B|-\rho(B)}.$$

Rearranging the summation signs gives

$$(10.3) \quad \sum_{B \subseteq E} (x-1)^{\rho(E)-\rho(B)} (y-1)^{|B|-\rho(B)} \sum_{\substack{A \\ B \subseteq A \subseteq E}} z^{|A|},$$

which equals

$$(10.4) \quad \sum_{B \subseteq E} (x-1)^{\rho(E)-\rho(B)} (y-1)^{|B|-\rho(B)} z^{|B|} (1+z)^{|E|-|B|}.$$

Rearranging terms yields

$$(10.5) \quad (1+z)^{|E|-\rho(E)} z^{\rho(E)} \sum_{B \subseteq E} \left((x-1)\frac{1+z}{z} \right)^{\rho(E)-\rho(B)} \left((y-1)\frac{z}{1+z} \right)^{|B|-\rho(B)},$$

which, by (2.3), gives the right-hand side of (10.1), as required. \square

Observe that for fixed $(x, y) \neq (1, 1)$ and variable $z \notin \{0, -1\}$, the right-hand side of (10.1) gives an expression for the Tutte polynomial along the special curve containing (x, y) . Equation (10.1) is used in sections 11 and 12 in the case $x = 0$.

Now the *flow polynomial* [21] of a matroid $N = (E, \mu)$ is defined to be

$$(10.6) \quad \text{Flow}(N; q) = (-1)^{|E|-\mu(E)} T(N; 0, 1-q).$$

So the left-hand side of (10.1), with $(x, y) = (0, 1-q)$, can be rewritten as

$$(10.7) \quad \sum_{A \subseteq E} z^{|A|} (-1)^{\rho(E)-|A|} \text{Flow}(M|A; q).$$

Substituting $v = -z$ and multiplying by $(-1)^{\rho(E)}$ gives

$$(10.8) \quad \sum_{A \subseteq E} v^{|A|} \text{Flow}(M|A; q) = v^{\rho(E)}(1 - v)^{|E| - \rho(E)} T \left(M; \frac{1}{v}, 1 + \frac{qv}{1 - v} \right),$$

so that the Tutte polynomial of M along the special curve H_q (or H_0^y for $q = 0$) is given, up to an easily computable factor, by the left-hand side of (10.8).

For a positive integer q and a graph G with $N = M(G)$, the flow polynomial counts the number of *nowhere-zero \mathbb{Z}_q -flows* of G , but here we actually need only a few very basic facts. First, $e \in E$ is a *loop* of N if it is contained in no circuit. In the graph case, e is also called a *bridge* or *cut-edge*. An edge with a degree-1 endpoint is certainly a coloop. If **loop** and **coloop** are single edge graphs (or matroids) consisting of a loop and coloop, respectively, then

$$(10.9) \quad \text{Flow}(\mathbf{loop}; q) = q - 1, \quad \text{Flow}(\mathbf{coloop}; q) = 0.$$

As a consequence of (2.4) and (10.6), if a matroid $N = (E, \mu)$ has connected components $(E_i : i \in I)$, then

$$(10.10) \quad \text{Flow}(N; x, y) = \prod_{i \in I} \text{Flow}(N|E_i; x, y).$$

Hence by (10.9) and (10.10), the following holds:

$$(10.11) \quad \text{If } N \text{ has a coloop, then for all } q, \text{Flow}(N; q) = 0.$$

Two graphs G, H are *homeomorphic*, denoted $G \simeq H$, if one can be obtained from the other, up to isomorphism, by subdividing edges and suppressing degree-2 vertices. Another basic property is the following:

$$(10.12) \quad \text{If } G \simeq H, \text{ then } \text{Flow}(G; q) = \text{Flow}(H; q).$$

Hence by (10.9), (10.10), (10.12), if $\dot{\cup}_{i=1}^p C_i$ denotes a graph that is a vertex disjoint union of p circuits (of any length), then

$$(10.13) \quad \text{Flow}(\dot{\cup}_{i=1}^p C_i; q) = (q - 1)^p.$$

Another type of useful identity arises from the radial construction and the results in Lemma 9.1. In some constructions and computations in sections 11 and 12, we will produce the quantity on the left-hand side of the equation

$$(10.14) \quad \sum_{A \subseteq E} a^{|A|} b^{|E| - |A|} d^{m_G(A, k)} = a^{\rho(E)} b^{|E| - \rho(E)} d^{k(|V| - \rho(E))} T \left(G; 1 + \frac{b}{a} d^k, 1 + \frac{a}{b} d^k \right),$$

which readers can verify by using (2.3) and (9.3). Setting $a = b = 1$, a simpler useful identity is

$$(10.15) \quad \sum_{A \subseteq E} d^{m_G(A, k)} = d^{k(|V| - \rho(E))} T(G; 1 + d^k, 1 + d^k).$$

11. The main constructions. In this section we examine two constructions and related combinatorial quantities that form the bulk of the proofs of Lemmas 12.3 and 12.6, which in turn are the largest steps toward proving Theorem 5.1.

Let k be a positive integer, and recall the $4 \cdot k$ -tile $\tilde{T}_\#^k$ with its blue and red edges as in Definition 8.2. For a positive integer m , we obtain a $4 \cdot k$ -tile $\tilde{T}_\#^{k,m}$ by subdividing every red edge of $\tilde{T}_\#^k$ into a path of m edges. The blue edges are not subdivided. See Figure 8.2 again, now viewing red (thin) lines as paths of m edges.

Let $G = (V, E)$ be a 2-connected plane directed graph with $n = |E|$ edges. Let k be a positive integer and let $m = 4k^2|E| = 4k^2n$. Construct the graph

$$(11.1) \quad G_\#^k = \Theta_G(T_e : e \in E), \text{ where } T_e = \tilde{T}_\#^{k,m}.$$

As with (9.1), this actually means each T_e is a copy of $\tilde{T}_\#^{k,m}$.

Let $E_\#^k$ be the edge set of $G_\#^k$. The edges of $G_\#^k$ are colored red and blue just as they are in each T_e . Recalling the discussion in section 8, we note that $G_\#^k$ has $2k(k+1)nm$ red edges and $m = 4k^2n$ blue edges, so that

$$(11.2) \quad |E_\#^k| = 2k(k+1)nm + 4k^2n = 2k(k+1)nm + m.$$

For a polynomial $p(v)$ let $\text{coeff}_{v^\delta}(f(v))$ denote the coefficient of v^δ in $p(v)$.

We will show that, for $q \in \mathbb{A}$, if we find the Tutte polynomial of $G_\#^k$ along curve H_q as per (10.8) and extract a particular coefficient, we will be able to use Lemma 9.1 and (10.15) to find $T(G; 1 + (q-1)^k, 1 + (q-1)^k)$. This is the key construction and computation for the reducibilities (12.1) and (12.2) in Lemma 12.3.

LEMMA 11.1. *Let $q \in \mathbb{A}$. Let k be a positive integer. Let $G = (V, E)$ and $n = |E|$ be as above. Let $m = 4k^2|E| = 4k^2n$. Construct $G_\#^k$ as above. Let $\delta = 2k(k+1)nm + 2k^2n = 2k(k+1)nm + m/2 = |E_\#^k| - m/2$. Then*

$$(11.3) \quad \text{coeff}_{v^\delta} \left(\sum_{B \subseteq E_\#^k} v^{|B|} \text{Flow}(G_\#^k|B; q) \right) = (q-1)^{k(|V| - \rho_G(E))} T(G; 1 + (q-1)^k, 1 + (q-1)^k).$$

Proof. The vertices in $G_\#^k$ all have degree 2 or 3. Consider the paths in $G_\#^k$ which join degree-3 vertices without going through any other degree-3 vertex. There are $m = 4k^2n$ degree-3 vertices and hence $6k^2n$ such paths, namely $4k^2n$ paths of length 1, $2k(k-1)n$ paths of length m , and $2kn$ paths of length $2m$. (The length- $2m$ paths arise from joining two length- m paths when “gluing” the $\tilde{T}_\#^{k,m}$ ’s together using the radial construction. These length- $2m$ paths could have been replaced by length- m paths, but it is not necessary and would be more inconvenient to define.) Any mention of length-1, length- m , or length- $2m$ paths during this proof refers only to the paths mentioned here. Note also that the length-1 paths are colored blue while the others are colored red.

Of course, the left-hand side of (11.3) can be written as

$$(11.4) \quad \sum_{\substack{B \subseteq E_\#^k \\ |B| = \delta}} \text{Flow}(G_\#^k|B; q),$$

and we can further eliminate terms for which $\text{Flow}(G_{\#}^k|B; q) = 0$, which by (10.11) certainly holds if $G_{\#}^k|B$ has a coloop, or in particular, a degree-1 vertex. Call B *essential* if $B \subseteq E_{\#}^k$, $|B| = \delta$, and $G_{\#}^k|B$ has no degree-1 vertex.

The graph $G_{\#}^k$ has been constructed so that any essential B must take a special form, as we now argue. Suppose B is essential. The choice of m and δ ensures that B must contain all the (red) paths of length m and $2m$ and exactly $2k^2n$ of the $4k^2n$ (blue) length-1 paths, since otherwise $G_{\#}^k|B$ would have a degree-1 vertex. In other words, B must contain all of the red edges and exactly half of the blue edges of $G_{\#}^k$.

In $G_{\#}^k$ every degree-3 vertex is an end of exactly two length-1 paths, and each length-1 path joins two degree-3 vertices. Since B contains all the length- m and length- $2m$ paths, each vertex with degree 3 in $G_{\#}^k$ has degree at least 1, and hence at least 2, in $G_{\#}^k|B$. Thus, each vertex with degree 3 in $G_{\#}^k$ is adjacent to at least one, and hence by the choice of δ , exactly one length-1 path in $G_{\#}^k|B$.

For each $e \in E$, in each copy T_e of $\tilde{T}_{\#}^{k,m}$ the $4k^2$ blue edges form a single circuit by condition (c) of Definition 8.2, and this condition will be crucial in what follows. Thus there are $n = |E|$ vertex disjoint blue circuits, one in each T_e . By the above discussion, every second edge of each blue circuit must be in an essential B . Thus for each $e \in E$ there are two choices, namely, the blue edges of T_e that are in B are precisely all the vertical edges or precisely all the horizontal edges. Therefore B must be of the form $B(A)$ for some $A \subseteq E$, where $B(A)$ contains all the red edges, $B(A)$ contains all the vertical blue edges in T_e for $e \in A$, and $B(A)$ contains all the horizontal blue edges in T_e for $e \notin A$.

Thus the quantity in (11.4) can now be written as

$$(11.5) \quad \sum_{A \subseteq E} \text{Flow}(G_{\#}^k|B(A); q).$$

Each $G_{\#}^k|B(A)$ simply consists of some number of vertex disjoint circuits that together meet every vertex of $G_{\#}^k$. If we examine the intersection $(G_{\#}^k|B(A)) \cap T_e$ regarded as a $4 \cdot k$ -tile, we find that each is essentially $T_{\{ \}$ or $T_{\{ \}$ (according to whether the vertical or horizontal blue edges are used), except that the paths in these $4 \cdot k$ -tiles are lengthened by the red edges. These lengthenings are inconsequential here, due to (10.12). For each $e \in E$, $A \subseteq E$,

$$(11.6) \quad (G_{\#}^k|B(A)) \cap T_e \simeq \begin{cases} T_{\{ \} }^k & \text{for } e \in A, \\ T_{\{ \} }^k & \text{for } e \in E - A, \end{cases}$$

where the \simeq signifies that the red edges merely lengthen the paths in each $T_{\{ \} }^k$ or $T_{\{ \} }^k$. Thus, recalling (9.2),

$$(11.7) \quad G_{\#}^k|B(A) \simeq G_A^k = \Theta_G(T_e : e \in E), \text{ where } T_e = \begin{cases} T_{\{ \} }^k & \text{for } e \in A, \\ T_{\{ \} }^k & \text{for } e \in E - A. \end{cases}$$

Considering Remark 1 we can apply Lemma 9.1 to find that $G_{\#}^k|B(A)$ consists of exactly $m_G(A, k) = k(|V| - 2\rho_G(A) + |A|)$ vertex disjoint circuits. Using (10.13), the quantity in (11.5) is

$$(11.8) \quad \sum_{A \subseteq E} (q - 1)^{m_G(A, k)},$$

and using (10.15) with $d = q - 1$ gives

$$(11.9) \quad (q - 1)^{k(|V| - \rho_G(E))} T(G; 1 + (q - 1)^k, 1 + (q - 1)^k)$$

as required. \square

The above is the key to proving Lemma 12.3, which is the major new idea beyond Theorems 4.2 and 4.3 that is needed to prove Theorem 5.1. However, the above idea still leaves a gap. Lemmas 12.2–12.5 deal with all points (x, y) except where $q = (x - 1)(y - 1) = 0$, leaving a gap, namely the special curves H_0^y and H_0^x (that is, the curves $y = 1$ and $x = 1$, respectively; see section 4). This gap requires an additional new idea which we now describe, leading to Lemma 11.2 below. A corresponding situation is also encountered in [19].

Recall the comments after (10.8). In this case, with $q = 0$, (12.1) in Lemma 12.3 gives only the useless reducibilities $\tau^0(\mathbb{P}\mathbb{G}, 0, 0) \leq_T \tau^1(\mathbb{P}\mathbb{G}, H_0^y)$ and $\tau^0(\mathbb{P}\mathbb{G}, 2, 2) \leq_T \tau^1(\mathbb{P}\mathbb{G}, H_0^y)$, depending on whether we use k odd or even. They are useless since the left-hand sides are functions in FP, giving no information about the right-hand sides.

Recall section 6. The trick is to modify the above ideas to show instead that $\tau_L^0(\mathbb{P}\mathbb{G}, 2, 2) \leq_T \tau^1(\mathbb{P}\mathbb{G}, H_0^y)$, considering (6.7). The modification is simply to use the construction similar to (11.1) except that one $4 \cdot k$ -tile, say T_g , will be chosen differently.

Let $G = (V, E)$ be a 2-connected plane directed graph with $n = |E|$ edges. Let $g \in E$. Fix $k = 2$ and let $m = 4k^2(n - 1) = 16(n - 1)$.

Define a $4 \cdot 2$ -tile T_\star (see Figure 11.1(a)) with the eight boundary vertices named as in section 7 (see also (7.1)), two internal vertices called vh and vt , and nine edges: one from vh to vt called the *central edge*, one each from vh to hr^i and lh^i , $i = 1, 2$, and one each from vt to rt^i and tl^i , $i = 1, 2$. Color these nine edges red.

Obtain the $4 \cdot 2$ -tile T_\star^m (see Figure 11.1(a) again) by subdividing each of the nine edges of T_\star into a path of m edges, so that T_\star^m has $9m$ edges. Again these $9m$ edges are red.

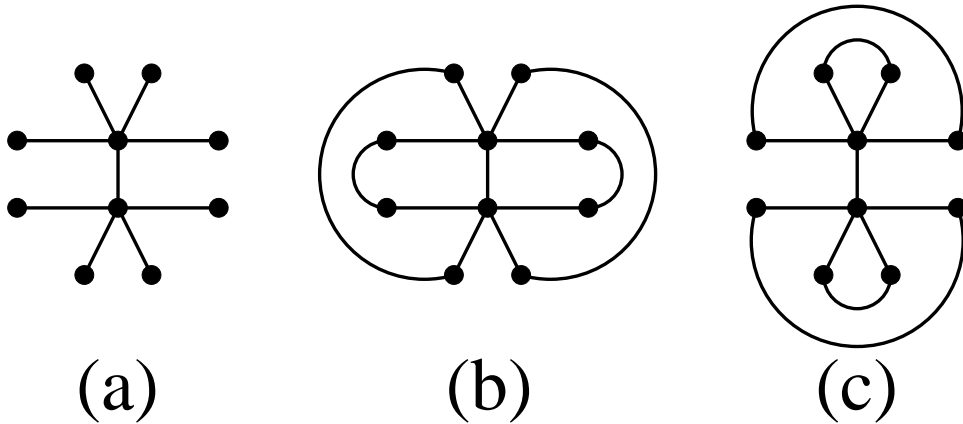


FIG. 11.1. (a) T_\star or T_\star^m where, respectively, each thin line depicts a red edge or a path of m red edges. (b) and (c) are graphs appearing in the proof of Lemma 11.2; see text.

Instead of $G_{\#\#}^k$ we will construct the graph $G_{\#\#}^{2gm}$:

$$(11.10) \quad G_{\#\#}^{2gm} = \Theta_G(T_e : e \in E), \text{ where } T_e = \begin{cases} \tilde{T}_{\#\#}^{2,m} & \text{for } e \neq g, \\ T_\star^m & \text{for } e = g. \end{cases}$$

Let $E_{\star\#\#}^{2gm}$ be the edge set of $G_{\star\#\#}^{2gm}$. The edges of $G_{\star\#\#}^{2gm}$ are colored red and blue just as they are in each T_e . This time we note that $G_{\star\#\#}^{2gm}$ has $(9 + 2k(k+1)(n-1))m = (9 + 12(n-1))m$ red edges and $m = 4k^2(n-1) = 16(n-1)$ blue edges, so that

$$(11.11) \quad |E_{\star\#\#}^{2gm}| = (9 + 12(n-1))m + 16(n-1) = (9 + 12(n-1))m + m.$$

In place of Lemma 11.1, we have Lemma 11.2 below.

LEMMA 11.2. *Set $q = 0$ and $k = 2$. Let $G = (V, E)$ and $n = |E|$ be as above. Let $g \in E$. Let $m = 16(n-1)$. Construct $G_{\star\#\#}^{2gm}$ as above. Let $\delta = (9 + 12(n-1))m + 8(n-1) = (9 + 12(n-1))m + m/2 = |E_{\star\#\#}^{2gm}| - m/2$. Then*

$$(11.12) \quad \text{coeff}_{v^\delta} \left(\sum_{B \subseteq E_{\star\#\#}^{2gm}} v^{|B|} \text{Flow}(G_{\star\#\#}^{2gm}|B; 0) \right) = 4T_L(G; 2, 2).$$

Proof. There are many similarities to the previous proof. Call B essential if $B \subseteq E_{\star\#\#}^{2gm}$, $|B| = \delta$, and $G_{\star\#\#}^{2gm}|B$ has no degree-1 vertex. By the choice of δ and the construction of $G_{\star\#\#}^{2gm}$ we find similarly as before that an essential B must be of the form $B(A)$ for some $A \subseteq E - g$, where $B(A)$ contains all the red edges (including all of T_g), $B(A)$ contains all the vertical blue edges in T_e for $e \in A$, and $B(A)$ contains all the horizontal blue edges in T_e for $e \in E - A - g$.

Similarly as before, the left-hand side of (11.12) can be written as

$$(11.13) \quad \sum_{A \subseteq E-g} \text{Flow}(G_{\star\#\#}^{2gm}|B(A); 0).$$

Also, similarly as before,

$$(11.14) \quad G_{\star\#\#}^{2gm}|B(A) \simeq \Theta_G(T_e : e \in E), \text{ where } T_e = \begin{cases} T_{\star}^k & \text{for } e \in A, \\ T_{\star}^k & \text{for } e \in E - A - g, \\ T_{\star}^m & \text{for } e = g. \end{cases}$$

Recall Lemma 9.1 and Remark 1. Either (1)–(7) all hold or (1′)–(7′) all hold, but not both. The graph $G_{\star\#\#}^{2gm}|B(A)$ will consist of the $4 \cdot 2$ -tile $T_g = T_{\star}^m$ with its eight boundary vertices pairwise joined by paths as in (6) or (6′), together with an even number $p = m_G(A, 2) - 2$ of vertex disjoint circuits. Considering (10.10) and (10.13) with $q = 0$ and p even, these vertex disjoint circuits contribute a factor of $(q-1)^p = 1$ to $\text{Flow}(G_{\star\#\#}^{2gm}|B(A); 0)$ and can be ignored.

If $A \in L(M(G), g)$, then in $G_{\star\#\#}^{2gm}|B(A)$ the boundary vertices of $T_g = T_{\star}^m$ are joined as in (6) of Lemma 9.1, giving a graph that is a subdivision of a graph consisting of just five parallel edges; see Figure 11.1(b). This graph has flow polynomial at $q = 0$ equaling 4.

If $A \notin L(M(G), g)$, then in $G_{\star\#\#}^{2gm}|B(A)$ the boundary vertices of $T_g = T_{\star}^m$ are joined as in (6′) of Lemma 9.1, giving a graph that has coloops, namely, those edges in the subdivided central edge; see Figure 11.1(c). This graph has flow polynomial at $q = 0$ equaling 0.

In summary, for $A \subseteq E - g$,

$$(11.15) \quad \text{Flow}(G_{\star\#\#}^{2gm}|B(A); 0) = \begin{cases} 4 & \text{for } A \in L(M(G), g), \\ 0 & \text{for } A \notin L(M(G), g). \end{cases}$$

Combining (6.6), (11.13), and (11.15), the result follows. \square

12. The remainder of the proofs. Theorem 5.1 is proved in a series of lemmas. Of course, significant portions of the proof are already presented in earlier sections.

Since by (2.4) any Tutte invariant of a graph is simply the product of that Tutte invariant of each of its blocks, we may restrict attention without further comment to 2-connected graphs.

LEMMA 12.1. *All of the functions mentioned in Theorem 5.1 are in $\overline{\#P}$. The function $\tau^2(\mathbb{P}\mathbb{G})$ is $\overline{\#P}$ -complete.*

Proof. By a simple observation in [17], $\tau^2(\mathbb{P}\mathbb{G}) \in \overline{\#P}$. Therefore all of the functions mentioned in Theorem 5.1 are in $\overline{\#P}$, by (4.5). Combining this with (6.7) and (6.10), the function $\tau^2(\mathbb{P}\mathbb{G})$ is $\overline{\#P}$ -complete. \square

LEMMA 12.2. *The functions claimed in Theorem 5.1 to be in FP are indeed so.*

Proof. If $(x - 1)(y - 1) = 1$ and $M = (E, \rho)$, then $T(M; x, y) = (x - 1)^{\rho(E)} y^{|E|}$ (in particular $T(M; 0, 0) = 0$) so that $\tau^1(\mathbb{P}\mathbb{G}, H_1)$ is in FP as are $\tau^0(\mathbb{P}\mathbb{G}, x, y)$ for all $(x, y) \in H_1$.

For a graph G the *Ising partition function* of G (see [4, 11, 9, 22]) is given (up to an easily computable factor) by the Tutte polynomial of $M(G)$ along the curve H_2 (see, for example, [22]). By [4, 11], this is polynomial time computable for planar graphs so that $\tau^1(\mathbb{P}\mathbb{G}, H_2)$ is in FP as are $\tau^0(\mathbb{P}\mathbb{G}, x, y)$ for all $(x, y) \in H_2$.

If $(x, y) \in \{(-1, -1), (0, -1), (-1, 0), (i, -i), (-i, i), (j, j^2), (j^2, j)\}$, then as shown in [18], $\tau^0(\mathbb{M}_q, x, y)$ is in FP, where \mathbb{M}_q is the class of matroids representable over the field of $q = (x - 1)(y - 1) = 4, 2, 2, 2, 2, 3, 3$ elements, respectively. Each of these classes contains $\mathbb{P}\mathbb{G}$ (and there is a polynomial time function sending each directed plane graph G to the appropriate representation of $M(G)$ for the above classes) so that $\tau^0(\mathbb{P}\mathbb{G}, x, y)$ is in FP for these (x, y) . Also $\tau^0(\mathbb{P}\mathbb{G}, 1, 1)$ is in FP since for any graph G , $T(M(G); 1, 1)$ is the number of spanning trees of G , and this can be computed in polynomial time using Kirchoff's determinantal formula. \square

By (4.5), it now suffices to show that $\tau^0(\mathbb{P}\mathbb{G}, x, y)$ is $\overline{\#P}$ -complete for every algebraic point (x, y) which is not special or on H_1 or H_2 (since every algebraic curve other than H_1 and H_2 contains such a point).

LEMMA 12.3. *If $q \in \mathbb{A}$, $q \neq 1$, and k is a positive integer, then*

$$(12.1) \quad \tau^0(\mathbb{P}\mathbb{G}, 1 + (q - 1)^k, 1 + (q - 1)^k) \leq_T \tau^1(\mathbb{P}\mathbb{G}, H_q).$$

If $q \in \mathbb{A}$ and $q - 1$ is not zero or a root of unity and “ $x = y$ ” is the nonspecial curve $\langle (x, y) | x, y \in \mathbb{A}, x = y \rangle$, then

$$(12.2) \quad \tau^1(\mathbb{P}\mathbb{G}, x = y) \leq_T \tau^1(\mathbb{P}\mathbb{G}, H_q).$$

Proof. For fixed k , the reduction of $\tau^0(\mathbb{P}\mathbb{G}, 1 + (q - 1)^k, 1 + (q - 1)^k)$ to $\tau^1(\mathbb{P}\mathbb{G}, H_q)$ is as follows. Suppose the input is the directed plane graph G with edge set E and $n = |E|$. Construct $G_{\#}^k$. Apply $\tau^1(\mathbb{P}\mathbb{G}, H_q)$ to $G_{\#}^k$ and multiply by a simple factor to get $v^{\rho_{G_{\#}^k}(E_{\#}^k)} (1 - v)^{|E_{\#}^k| - \rho_{G_{\#}^k}(E_{\#}^k)} T(G_{\#}^k; \frac{1}{v}, 1 + \frac{qv}{1-v})$, which by (10.8) equals $\sum_{B \subseteq E_{\#}^k} v^{|B|} \text{Flow}(G_{\#}^k | B; q)$. With δ as in Lemma 11.1, take the coefficient of v^{δ} and divide by $(q - 1)^{k(|V| - \rho_G(E))}$ (which is nonzero since $q \neq 1$) to obtain $T(G; 1 + (q - 1)^k, 1 + (q - 1)^k)$ by Lemma 11.1, as required.

Note that all of the above can be done in time polynomial in both n and k . Note also that the degree of $T(G; x, x)$ is at most $2n$. For the second reducibility (12.2) do as above for $k = 1, \dots, 2n + 1$ (in time polynomial in n) to obtain $T(G; x, x)$ at $2n + 1$ distinct points, namely, $x = 1 + (q - 1)^k$ for $k = 1, \dots, 2n + 1$. Here we

have used the assumption that $q - 1$ is not zero or a root of unity. This one variable polynomial, with degree at most $2n$, can then be obtained in polynomial time by Lagrange interpolation. \square

LEMMA 12.4. *If $q, x, y \in \mathbb{A}$, $q - 1$ is not zero or a root of unity and $(x, y) \in H_q$, where (x, y) is not special, then $\tau^0(\mathbb{P}\mathbb{G}, x, y)$ is $\overline{\#P}$ -complete.*

Proof. By Lemma 12.1 $\tau^2(\mathbb{P}\mathbb{G})$ is $\overline{\#P}$ -complete. By Theorem 4.2, $\tau^2(\mathbb{P}\mathbb{G}) \leq_T \tau^1(\mathbb{P}\mathbb{G}, x = y)$. By Lemma 12.3, $\tau^1(\mathbb{P}\mathbb{G}, x = y) \leq_T \tau^1(\mathbb{P}\mathbb{G}, H_q)$. By Theorem 4.3, $\tau^1(\mathbb{P}\mathbb{G}, H_q) \leq_T \tau^0(\mathbb{P}\mathbb{G}, x, y)$. Therefore $\tau^0(\mathbb{P}\mathbb{G}, x, y)$ is $\overline{\#P}$ -complete. \square

LEMMA 12.5. *If $q, x, y \in \mathbb{A}$, $q \notin \{0, 1, 2\}$, $q - 1$ is a root of unity, and $(x, y) \in H_q$, where (x, y) is not special, then $\tau(\mathbb{P}\mathbb{G}, x, y)$ is $\overline{\#P}$ -complete.*

Proof. Since $q - 1 \neq 0$, then by Lemma 12.3 $\tau^0(\mathbb{P}\mathbb{G}, 1 + (q - 1)^k, 1 + (q - 1)^k) \leq_T \tau^1(\mathbb{P}\mathbb{G}, H_q)$ for each fixed $k \in \{1, 2, \dots\}$ and by Lemma 12.4, the left-hand side is $\overline{\#P}$ -complete unless [i] $(q - 1)^{2k} - 1$ is zero or a root of unity (since $(1 + (q - 1)^k, 1 + (q - 1)^k) \in H_{(q-1)^{2k}}$), or [ii] $(1 + (q - 1)^k, 1 + (q - 1)^k)$ is a special point. In case [ii], $(q - 1)^k = 0, -1, -2$. Now $(q - 1)^k = 0, -2$ contradicts $q - 1$ being a root of unity, and $(q - 1)^k = -1$ implies $(q - 1)^{2k} - 1$ is zero, so that both cases [i] and [ii] are covered by the following assumption. Suppose that $(q - 1)^{2k} - 1$ is zero or a root of unity for all $k \in \{1, 2, \dots\}$. Since $(q - 1)^{2k}$ is a root of unity, $(q - 1)^{2k}$ must be one of $1, -j, -j^2$ for all $k \in \{1, 2, \dots\}$. But if $(q - 1)^2 \in \{-j, -j^2\}$, then $(q - 1)^6 = -1 \notin \{1, -j, -j^2\}$, and if $(q - 1)^2 = 1$, then $q \in \{0, 2\}$, a contradiction. Thus $\tau^0(\mathbb{P}\mathbb{G}, 1 + (q - 1)^k, 1 + (q - 1)^k)$, and hence $\tau^1(\mathbb{P}\mathbb{G}, H_q)$, is $\overline{\#P}$ -complete. By Theorem 4.3, $\tau^1(\mathbb{P}\mathbb{G}, H_q) \leq_T \tau^0(\mathbb{P}\mathbb{G}, x, y)$ and since the left-hand side is $\overline{\#P}$ -complete, so is the right-hand side. \square

Only the case $(x, y) \in H_0$ remains.

LEMMA 12.6. $\tau_L^0(\mathbb{P}\mathbb{G}, 2, 2) \leq_T \tau^1(\mathbb{P}\mathbb{G}, H_0^y)$.

Proof. The reduction of $\tau_L^0(\mathbb{P}\mathbb{G}, 2, 2)$ to $\tau^1(\mathbb{P}\mathbb{G}, H_0^y)$ is as follows. Suppose the input is the directed plane graph G with edge set E and $n = |E|$. Construct $G_{*\#}^{2gm}$. Abbreviate $G_{*\#}^{2gm}$ and $E_{*\#}^{2gm}$ to G' and E' , respectively.

Apply $\tau^1(\mathbb{P}\mathbb{G}, H_0^y)$ to G' , and multiply by a simple factor to get $v^{\rho_{G'}(E')} (1 - v)^{|E'| - \rho_{G'}(E')} T(G'; \frac{1}{v}, 1)$, which by (10.8) with $q = 0$ equals $\sum_{B \subseteq E'} v^{|B|} \text{Flow}(G'|B; q)$.

With δ as in Lemma 11.2, take the coefficient of v^δ and divide by 4 to obtain $T_L(G; 2, 2)$ by Lemma 11.2, as required. \square

LEMMA 12.7. *If $x, y \in \mathbb{A}$, $(x, y) \in H_0 - \{(1, 1)\}$ (so that $x = 1$ or $y = 1$ but $(x, y) \neq (1, 1)$), then $\tau^0(\mathbb{P}\mathbb{G}, x, y)$ is $\overline{\#P}$ -complete.*

Proof. By Lemma 12.6, $\tau_L^0(\mathbb{P}\mathbb{G}, 2, 2) \leq_T \tau(\mathbb{P}\mathbb{G}, H_0^y)$. By Theorem 4.3, $\tau(\mathbb{P}\mathbb{G}, H_0^y) \leq_T \tau(\mathbb{P}\mathbb{G}, x, 1)$ for all $x \in \mathbb{A} - \{1\}$. Since $\tau_L^0(\mathbb{P}\mathbb{G}, 2, 2)$ is $\overline{\#P}$ -complete by (6.7), so is $\tau^0(\mathbb{P}\mathbb{G}, x, 1)$. Also $\tau^0(\mathbb{P}\mathbb{G}, x, 1) \leq_T \tau^0(\mathbb{P}\mathbb{G}, 1, x)$ (and vice versa) since for any plane graph G , its plane dual G^* can be constructed in polynomial time and $T(M(G^*); x, y) = T(M(G); y, x)$ (see [21]). Hence $\tau^0(\mathbb{P}\mathbb{G}, 1, y)$ is $\overline{\#P}$ -complete for all $y \in \mathbb{A} - \{1\}$. \square

We are finally in a position to prove Theorem 5.1.

Proof. All the cases have been covered, and Theorem 5.1 is proved. \square

13. Conclusions. For evaluations of the Tutte polynomial at a fixed algebraic point or curve, the computational complexity is the same for planar graphs (Theorem 5.1) as for graphs (Proposition 4.4), except along the curve H_2 (the ‘‘Ising model’’) and at the nonspecial points on H_2 . It is natural to consider the computational complexity of these evaluations for smaller (interesting) classes of graphs. For the class $3\mathbb{P}\mathbb{G}$, say, of planar graphs with vertex degree at most 3, the result is the

same as Theorem 5.1 (substituting $3\mathbb{P}\mathbb{G}$ for $\mathbb{P}\mathbb{G}$) except that $\tau^0(3\mathbb{P}\mathbb{G}, 0, -2)$ (which counts 3-flows) is polynomial time computable [20].

It is also natural to examine the computational complexity of Tutte invariants for larger classes of matroids. This examination leads to connections between computational complexity, representability over finite fields, and uniqueness of bicycle dimension; see [18].

The Tutte polynomial of planar graphs is closely related to the Homfly and Kauffman polynomials for knots and links. Theorem 5.1 forms part of the argument determining the complexity of computing Homfly and Kauffman invariants; see [19].

Acknowledgment. I would like to thank my wife, Jilyana Cazaran, for help with the drawing of figures, typesetting, and proofreading.

REFERENCES

- [1] T. BRYLAWSKI, *The Tutte polynomial. I. General theory*, in *Matroid Theory and its Applications*, Liguori, Naples, Italy, 1982, pp. 125–275.
- [2] T. H. BRYLAWSKI, *A decomposition for combinatorial geometries*, *Trans. Amer. Math. Soc.*, 171 (1972), pp. 235–282.
- [3] H. H. CRAPO, *The Tutte polynomial*, *Aequationes Math.*, 3 (1969), pp. 211–229.
- [4] M. E. FISHER, *On the Dimer solution of planar Ising models*, *J. Math. Phys.*, 7 (1966), pp. 1776–1781.
- [5] M. R. GAREY AND D. S. JOHNSON, *Computers and intractability. A Guide to the Theory of NP-completeness*, A Series of Books in the Mathematical Sciences, W. H. Freeman, San Francisco, 1979.
- [6] M. GRÖTSCHEL, L. LOVÁSZ, AND A. SCHRIJVER, *Geometric Algorithms and Combinatorial Optimization*, Algorithms and Combinatorics: Study and Research Texts 2, Springer-Verlag, Berlin, 1988.
- [7] F. JAEGER, *Tutte polynomials and link polynomials*, *Proc. Amer. Math. Soc.*, 103 (1988), pp. 647–654.
- [8] F. JAEGER, D. L. VERTIGAN, AND D. J. A. WELSH, *On the computational complexity of the Jones and Tutte polynomials*, *Math. Proc. Cambridge Philos. Soc.*, 108 (1990), pp. 35–53.
- [9] M. JERRUM, *Two-dimensional Monomer-Dimer systems are computationally intractable*, *J. Statist. Phys.*, 48 (1987), pp. 121–134.
- [10] M. JERRUM, *private communication*, 1989.
- [11] P. W. KASTELEYN, *Graph theory and crystal physics*, in *Graph Theory and Theoretical Physics*, Academic Press, London, 1967, pp. 43–110.
- [12] J. G. OXLEY, *Matroid Theory*, Oxford Science Publications, The Clarendon Press, Oxford University Press, New York, 1992.
- [13] J. S. PROVAN, *The complexity of reliability computations in planar and acyclic graphs*, *SIAM J. Comput.*, 15 (1986), pp. 694–702.
- [14] M. B. THISTLETHWAITE, *A spanning tree expansion of the Jones polynomial*, *Topology*, 26 (1987), pp. 297–309.
- [15] W. T. TUTTE, *A ring in graph theory*, *Proc. Cambridge Philos. Soc.*, 43 (1947), pp. 26–40.
- [16] L. G. VALIANT, *The complexity of enumeration and reliability problems*, *SIAM J. Comput.*, 8 (1979), pp. 410–421.
- [17] D. VERTIGAN, *The Computational Complexity of Tutte, Jones, Homfly, and Kauffman Invariants*, DPhil thesis, Oxford University, Oxford, England, 1991.
- [18] D. VERTIGAN, *Bicycle dimension and special points of the Tutte polynomial*, *J. Combin. Theory Ser. B*, 74 (1998), pp. 378–396.
- [19] D. VERTIGAN, *The Computational Complexity of Homfly and Kauffman Invariants for Links*, manuscript.
- [20] D. VERTIGAN, *The Computational Complexity of Tutte Invariants for Cubic Planar Graphs*, manuscript.
- [21] D. J. A. WELSH, *Matroid Theory*, London Math. Soc. Monogr. 8, Academic Press [Harcourt Brace Jovanovich], London, 1976.
- [22] D. J. A. WELSH, *Complexity: Knots, Colourings and Counting*, London Math. Soc. Lecture Note Ser. 186, Cambridge University Press, Cambridge, UK, 1993.
- [23] H. WHITNEY, *A logical expansion in mathematics*, *Bull. Amer. Math. Soc.*, 38 (1932), pp. 572–579.

A POLYNOMIAL TIME APPROXIMATION SCHEME FOR THE MULTIPLE KNAPSACK PROBLEM*

CHANDRA CHEKURI[†] AND SANJEEV KHANNA[‡]

Abstract. The *multiple knapsack problem* (MKP) is a natural and well-known generalization of the single knapsack problem and is defined as follows. We are given a set of n items and m bins (knapsacks) such that each item i has a profit $p(i)$ and a size $s(i)$, and each bin j has a capacity $c(j)$. The goal is to find a subset of items of maximum profit such that they have a feasible packing in the bins. MKP is a special case of the *generalized assignment problem* (GAP) where the profit and the size of an item can vary based on the specific bin that it is assigned to. GAP is APX-hard and a 2-approximation, for it is implicit in the work of Shmoys and Tardos [*Math. Program. A*, 62 (1993), pp. 461–474], and thus far, this was also the best known approximation for MKP. The main result of this paper is a polynomial time approximation scheme (PTAS) for MKP.

Apart from its inherent theoretical interest as a common generalization of the well-studied knapsack and bin packing problems, it appears to be the strongest special case of GAP that is not APX-hard. We substantiate this by showing that slight generalizations of MKP are APX-hard. Thus our results help demarcate the boundary at which instances of GAP become APX-hard. An interesting aspect of our approach is a PTAS-preserving reduction from an arbitrary instance of MKP to an instance with $O(\log n)$ distinct sizes and profits.

Key words. multiple knapsack problem, generalized assignment problem, polynomial time approximation scheme, approximation algorithm

AMS subject classifications. 68Q17, 68Q25, 68W25

DOI. 10.1137/S0097539700382820

1. Introduction. We study the following natural generalization of the classical knapsack problem.

Multiple knapsack problem (MKP).

INSTANCE: A pair $(\mathcal{B}, \mathcal{S})$ where \mathcal{B} is a set of m bins (knapsacks) and \mathcal{S} is a set of n items. Each bin $j \in \mathcal{B}$ has a capacity $c(j)$, and each item i has a size $s(i)$ and a profit $p(i)$.

OBJECTIVE: Find a subset $U \subseteq \mathcal{S}$ of maximum profit such that U has a feasible packing in \mathcal{B} .

The decision version of MKP is a generalization of the decision versions of both the knapsack and bin packing problems and is strongly NP-complete. Moreover, it is an important special case of the *generalized assignment problem* where both the size and the profit of an item are a function of the bin.

*Generalized assignment problem (GAP).*¹

INSTANCE: A pair $(\mathcal{B}, \mathcal{S})$ where \mathcal{B} is a set of m bins (knapsacks) and \mathcal{S} is a set of n items. Each bin $j \in \mathcal{B}$ has a capacity $c(j)$, and for each item i and bin j , we are given a size $s(i, j)$ and a profit $p(i, j)$.

*Received by the editors December 22, 2000; accepted for publication (in revised form) July 7, 2005; published electronically February 3, 2006. A preliminary version of this paper appeared in *Proceedings of the 11th Annual ACM-SIAM Symposium on Discrete Algorithms*, 2000, pp. 213–222.

<http://www.siam.org/journals/sicomp/35-3/38282.html>

[†]Bell Labs, 600 Mountain Ave., Murray Hill, NJ 07974 (chekuri@research.bell-labs.com).

[‡]Department of Computer and Information Science, University of Pennsylvania, Philadelphia, PA 19104 (sanjeev@cis.upenn.edu). This work was done while the author was at Bell Labs.

¹GAP has also been defined in the literature as a (closely related) minimization problem (see [28]). In this paper, following [26], we refer to the maximization version of the problem as GAP and to the minimization version as Min GAP.

OBJECTIVE: Find a subset $U \subseteq \mathcal{S}$ that has a feasible packing in \mathcal{B} and maximizes the profit of the packing.

GAP and its restrictions capture several fundamental optimization problems and have many practical applications in computer science, operations research, and related disciplines. The special case of MKP with uniform sizes and profits is also important; the book on knapsack variants by Martello and Toth [26] has a chapter devoted to MKP. Also referred to as the loading problem in operations research, it models the problem of loading items into containers of different capacities such that container capacities are not violated. In many practical settings items could be more complex geometric objects; however, the one-dimensional case (MKP) is useful in its own right and has been investigated extensively [14, 11, 12, 23, 24, 25, 4].

Knapsack, bin packing, and related problems have attracted much theoretical attention for their simplicity and elegance, and their study has been instrumental in the development of the theory of approximation algorithms. Though knapsack and bin packing have a fully polynomial-time approximation scheme (FPTAS; asymptotic for bin packing), GAP, a strong generalization of both, is APX-hard, and only a 2-approximation exists. In fact, some very special cases of GAP can be shown to be APX-hard. In particular we can show that for arbitrarily small $\delta > 0$ (which can even be a function of n) the problem remains APX-hard on the following very restricted set of instances: bin capacities are identical, and for each item i and machine j , $p(i, j) = 1$, and $s(i, j) = 1$ or $s(i, j) = 1 + \delta$. The complementary case, where item sizes do not vary across bins but profits do, can also be shown to be APX-hard for a similar restricted setting. In light of this, it is particularly interesting to understand the complexity of MKP where profits and sizes of an item are independent of the bin, but the item sizes and profits as well as bin capacities may take arbitrary values. Establishing a PTAS shows a very fine separation between cases that are APX-hard and those that have a PTAS. Until now, the best known approximation ratio for MKP was a factor of 2 derived from the approximation for GAP.

Results. In this paper we resolve the approximability of MKP by obtaining a PTAS for it. It can be easily shown via a reduction from the Partition problem that MKP does not admit an FPTAS even if $m = 2$ (see Proposition 2.1). A special case of MKP is when all bin capacities are equal. It is relatively straightforward to obtain a PTAS for this case using ideas from approximation schemes for knapsack and bin packing [13, 3, 17]. However, the problem with *different* bin capacities is more challenging. Our paper contains two new technical ideas. Our first idea concerns the set of items to be packed in a knapsack instance. We show how to guess, in polynomial time, *almost all* the items that are packed by an optimal solution. More precisely, we can identify a polynomial number of subsets such that one of the subsets has a *feasible* packing and profit at least $(1 - \epsilon)\text{OPT}$. This is in contrast to earlier schemes for variants of knapsack [13, 1, 7], where only the $1/\epsilon$ most profitable items are guessed. An easy corollary of our strategy is a PTAS for the identical bin capacity case, the details of which we point out later. Even with the knowledge of the right subsets, the problem remains nontrivial since we need to verify for each subset if it has a feasible packing. Checking for feasibility is of course at least as hard as bin packing. To get around this difficulty we make crucial use of additional properties satisfied by the subsets that we guess. In particular, we show that each subset can be transformed such that the number of *distinct* size values of the items in the subset is $O(\epsilon^{-2} \log n)$. An immediate consequence of this is a dynamic programming-based quasi-polynomial time algorithm to pack all of the items into bins. Our second set of ideas shows that we can exploit the restriction on the number of distinct sizes

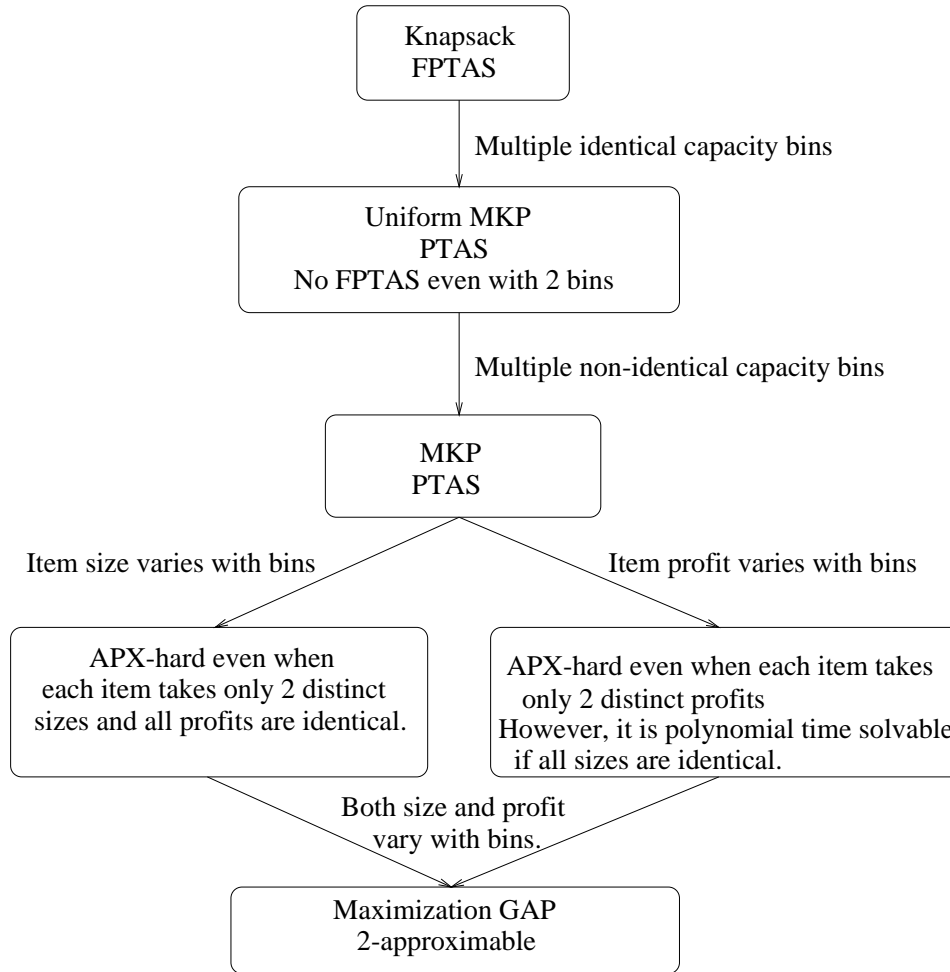


FIG. 1.1. Complexity of various restrictions of GAP.

to pack, in *polynomial* time, a subset of the item set that has at least a $(1 - \epsilon)$ fraction of the profit. Approximation schemes for number problems are usually based on rounding instances to have a fixed number of distinct values. In contrast, MKP appears to require a logarithmic number of values. We believe that our techniques to handle logarithmic number of distinct values will find other applications. Figure 1.1 summarizes the approximability of various restrictions of GAP.

Related work. MKP is closely related to knapsack, bin packing, and GAP. A very efficient FPTAS exists for the knapsack problem; Lawler's [19], based on ideas from [13], achieves a running time of $O(n \log 1/\epsilon + 1/\epsilon^4)$ for a $(1 + \epsilon)$ approximation. An asymptotic FPTAS is known for bin packing [3, 17]. Kellerer [18] has independently developed a PTAS for the special case of the MKP where all bins have *identical* capacity. As mentioned earlier, this case is much simpler than the general case and falls out as a consequence of our first idea. We defined the generalized assignment problem as a maximization problem. This is natural when we relate it to the knapsack problem (see [26]). There is also a minimization version, which we refer to as Min GAP (also known as the cost assignment problem), where the objective is to assign all the

items while *minimizing* the sum of the costs of assigning items to bins. In this version, item i when assigned to bin j incurs a *cost* $w(i, j)$ instead of obtaining a profit $p(i, j)$. Even without costs, deciding the feasibility of assigning all items without violating the capacity constraints is an NP-complete problem; therefore, capacity constraints need to be relaxed. An (α, β) bicriteria approximation algorithm for Min GAP is one that gives a solution with cost at most αC and with bin capacities violated by a factor of at most β , where C is the cost of an optimal solution that does not violate any capacity constraints. The work of Lin and Vitter [22] yields a $(1 + \epsilon, 2 + 1/\epsilon)$ bicriteria approximation for Min GAP. Shmoys and Tardos [28], building on the work of Lenstra, Shmoys, and Tardos [21], give an improved $(1, 2)$ bicriteria approximation. Implicit in this approximation is also a 2-approximation for the profit maximization version which we sketch later. Lenstra, Shmoys, and Tardos [21] also show that it is NP-hard to obtain a bicriteria approximation of the form $(1, \beta)$ for any $\beta < 3/2$. The hardness relies on an NP-completeness reduction from the decision version of the 3-dimensional matching problem. Our APX-hardness for the maximization version, mentioned earlier, is based on a similar reduction but instead relies on APX-hardness of the optimization version of the 3-dimensional matching problem [16].

MKP is also related to two variants of variable-size bin packing. In the first variant we are given a set of items and set of bin capacities \mathcal{C} . The objective is to find a feasible packing of items using bins with capacities restricted to be from \mathcal{C} so as to minimize the sum of the capacities of the bins used. A PTAS for this problem was provided by Murgolo [27]. The second variant is based on a connection to multiprocessor scheduling on uniformly related machines [20]. The objective is to assign a set of jobs with given processing times to machines with different speeds so as to minimize the makespan of the schedule. Hochbaum and Shmoys [10] gave a PTAS for this problem using a *dual*-based approach where they convert the scheduling problem into the following bin packing problem. Given items of different sizes and bins of different capacities, find a packing of all the items into the bins such that maximum relative violation of the capacity of any bin is minimized. Bicriteria approximations, where both capacity and profit can be approximated simultaneously, have been studied for several problems (Min GAP being an example mentioned above), and it is usually the case that relaxing both makes the task of approximation somewhat easier. In particular, relaxing the capacity constraints allows rounding of item sizes into a small number of distinct size values. In MKP, the constraint on the bin capacities and the constraint on the number of bins are both inviolable, and this makes the problem harder.

Organization. Section 2 describes our PTAS for MKP. In section 3, we show that GAP is APX-hard on very restricted classes of instances. We also indicate here a 2-approximation for GAP. In section 4, we discuss a natural greedy algorithm for MKP and show that it gives a $(2 + \epsilon)$ -approximation even when item sizes vary with bins.

2. A PTAS for the multiple knapsack problem. We first show that MKP does not admit an FPTAS even for $m = 2$.

PROPOSITION 2.1. *If MKP with two identical bins has an FPTAS, then the Partition problem can be solved in polynomial time. Hence there is no FPTAS for MKP even with $m = 2$ unless $P = NP$.*

Proof. An instance of the Partition problem consists of $2n$ numbers a_1, a_2, \dots, a_{2n} , and the goal is to decide if the numbers can be partitioned into two sets S_1 and S_2 such that the sum of numbers in each set add up to exactly $A = \frac{1}{2} \sum_{i=1}^{2n} a_i$. We can

reduce the Partition problem to MKP with two bins as follows. We set the capacity of the bins to be A . We have $2n$ items, one for each number in the Partition problem: the size of item i is a_i and the profit of item i is 1. If the Partition problem has a solution, the profit of an optimum solution to the corresponding MKP problem is $2n$; otherwise it is at most $2n - 1$. Thus, an FPTAS for MKP can be used to distinguish between these two situations in polynomial time. \square

We start with a remark on guessing. When we *guess* a quantity in polynomial time, we mean that we can identify, in polynomial time, a polynomial size set of values among which the correct value of the desired quantity resides. Coupled with the guessing procedure is a polynomial time checking procedure which can verify whether a feasible solution with a given value exists. We can run the checking procedure with each of the values in the guessed set and will be guaranteed to obtain a solution with the correct value. We will be using this standard idea several times in this section and implicitly assume that the above procedure is invoked to complete the algorithm.

We denote by OPT the value of an optimal solution to the given instance. Given a set Y of items, we use $p(Y)$ to denote $\sum_{y \in Y} p(y)$. The set of integers $0, 1, \dots, k$ is denoted by $[k]$. We will assume throughout this section that $\epsilon < 1$; when $\epsilon \geq 1$ we can use the 2-approximation for GAP from section 3. In the rest of the paper we assume, for simplicity of notation, that $1/\epsilon$ and $\ln n$ are integers. Further, we also assume that $\epsilon > 1/n$, for otherwise we can use an exponential time algorithm to solve the problem exactly. In several places in the paper, to simplify expressions, we use the inequality $\ln(1 + \epsilon) \geq \epsilon - \epsilon^2/2 \geq \epsilon/2$.

Our problem is related to both the knapsack problem and the bin packing problem, and some ideas used in approximation schemes for those problems will be useful to us. Our approximation scheme conceptually has the following two steps.

1. Guessing Items: Identify a set of items $U \subseteq \mathcal{S}$ such that $p(U) \geq (1 - \epsilon)\text{OPT}$ and U has a feasible packing in \mathcal{B} .
2. Packing Items: Given a set U of items that has a feasible packing in \mathcal{B} , find a feasible packing for a set $U' \subseteq U$ such that $p(U') \geq (1 - \epsilon)p(U)$.

The overall scheme is more involved since there is interaction between the two steps. The guessed items have some additional properties that are exploited in the packing step. We observe that both of the above steps require new ideas. For the regular single knapsack problem, the second step is trivial once we accomplish the first step. This is, however, not the case for MKP. Before we proceed with the details we show how our guessing step immediately gives a PTAS for the identical bin capacity case.

2.1. MKP with identical bin capacities. Suppose we can guess an item set as in our first step above. We show that the packing step is very simple if the bin capacities are identical. There are two cases to consider, depending on whether m , the number of bins, is less than or equal to $1/\epsilon$ or not. If $m \leq 1/\epsilon$, the number of bins can be treated as a constant, and a PTAS for this case exists even for instances of GAP (implicit in earlier work [7]). Now suppose $m > 1/\epsilon$. Bin packing has an asymptotic PTAS. In particular, there is an algorithm [3] that packs the items into $(1 + \epsilon)\text{OPT} + 1$ bins in polynomial time for any fixed $\epsilon > 0$. We can thus use this algorithm to pack *all* the guessed items using at most $(1 + \epsilon)m + 1$ bins. We find a feasible solution by simply picking the m largest profit bins and discarding the rest along with their items. Here we use the fact that $m\epsilon \geq 1$ and that the bins are identical. It is easily seen that we get a $(1 + O(\epsilon))$ approximation. We note that a different PTAS, one that does not rely on our guessing step, can be obtained for this case by directly adapting

the ideas used in approximation schemes for bin packing. The trick of using extra bins does not have a simple analogue when bin capacities are different and we need more sophisticated ideas for the general case.

2.2. Guessing items. Consider the case when all items have the same profit; without loss of generality assume it is 1. Thus the objective is to pack as many items as possible. For this case, it is easily seen that OPT is an integer in $[n]$. Further, given a guess \mathcal{O} for OPT , we can pick the *smallest* (in size) \mathcal{O} items to pack. Thus knowing \mathcal{O} allowed us to fix the item set as well. Therefore there are only a polynomial number of guesses for the set of items to pack. In the following we build on this useful insight.

Let p_{\max} denote the largest value among item profits. For the general case the first step involves massaging the given instance into a more structured one that has few distinct profits. This is accomplished as follows.

1. Guess a value \mathcal{O} such that $\max\{p_{\max}, (1 - \epsilon)\text{OPT}\} \leq \mathcal{O} \leq \text{OPT}$ and discard all items y where $p(y) < \epsilon\mathcal{O}/n$.
2. Divide all profits by $\epsilon\mathcal{O}/n$ such that after scaling each profit is at most n/ϵ .
3. Round *down* the profits of items to the nearest power of $(1 + \epsilon)$.

It is easily seen that only an $O(\epsilon)$ fraction of the optimal profit is lost by our transformation. Since we do not know OPT , we need to establish an upper bound on the number of values of \mathcal{O} that we will try out. We make use of the following easy bounds on OPT : $p_{\max} \leq \text{OPT} \leq n \cdot p_{\max}$.

Therefore, one of the values in $\{p_{\max} \cdot (1 + \epsilon)^i \mid 0 \leq i \leq 2\epsilon^{-1} \ln n\}$ is guaranteed to satisfy the desired properties for \mathcal{O} . Summarizing, we obtain the following lemma.

LEMMA 2.2. *Given an instance $I = (\mathcal{B}, \mathcal{S})$ with n items and a value \mathcal{O} such that $(1 - \epsilon)\text{OPT}(I) \leq \mathcal{O} \leq \text{OPT}(I)$, we can obtain in polynomial time another instance $I' = (\mathcal{B}, \mathcal{S}')$ such that*

- $\mathcal{S}' \subseteq \mathcal{S}$;
- for every $y \in \mathcal{S}'$, $p(y) = (1 + \epsilon)^i$ for some $i \in [4\epsilon^{-1} \ln n]$;
- $(1 - \epsilon)\text{OPT}(I) \leq \frac{n}{\epsilon\mathcal{O}}\text{OPT}(I') \leq \text{OPT}(I)$.

For the bound in the second item above we upper bound n/ϵ by n^2 . The above lemma allows us to work with instances with $O(\epsilon^{-1} \ln n)$ distinct profits. We now show how we can use this information to guess the items to pack. Let $h \leq 4\epsilon^{-1} \ln n + 1$ be the number of distinct profits in our new instance. We partition \mathcal{S} into h sets S_1, \dots, S_h with items in each set having the same profit. Let U be the items chosen in some optimal solution and let $U_i = S_i \cap U$. Recall that we have an estimate \mathcal{O} of the optimal value. If $p(U_i) \leq \epsilon\mathcal{O}/h$ for some i , we ignore the set S_i ; no significant profit is lost. Hence we can assume that $\epsilon\mathcal{O}/h \leq p(U_i) \leq \mathcal{O}$ and approximately guess the value $p(U_i)$ for $1 \leq i \leq h$. More precisely, for each i we guess a value $k_i \in [h/\epsilon^2]$ such that $k_i(\epsilon^2\mathcal{O}/h) \leq p(U_i) \leq (k_i + 1)(\epsilon^2\mathcal{O}/h)$.

A naive way of guessing the values k_1, \dots, k_h requires $n^{\Omega(\ln n/\epsilon^2)}$ time. We first show how the numbers k_i enable us to identify the items to pack and then show how the values k_1, \dots, k_h can in fact be guessed in polynomial time. Let a_i denote the profit of an item in S_i . Consider an index i such that $a_i \leq \epsilon\mathcal{O}/h$. Given the value k_i we order the items in S_i in increasing size values and pick the largest number of items from this ordered set whose cumulative profit does not exceed $k_i(\epsilon^2\mathcal{O}/h)$. If $a_i > \epsilon\mathcal{O}/h$ we pick the smallest number of items, again in order of increasing size, whose cumulative profit exceeds $k_i(\epsilon^2\mathcal{O}/h)$. The asymmetry is for technical reasons. The choice of items is thus completely determined by the choice of the k_i . For a tuple of values k_1, \dots, k_h , let $U(k_1, \dots, k_h)$ denote the set of items picked as described above.

LEMMA 2.3. *There exists a valid tuple (k_1, \dots, k_h) with each $k_i \in [h/\epsilon^2]$ such that $U(k_1, \dots, k_h)$ has a feasible packing in \mathcal{B} and $p(U(k_1, \dots, k_h)) \geq (1 - \epsilon)\mathcal{O}$.*

Proof. Let U be the items in some optimal solution and let $U_i = S_i \cap U$. Define k_i to be $\lfloor p(U_i)h/(\epsilon^2\mathcal{O}) \rfloor$. The tuple so obtained satisfies the required properties. \square

As remarked earlier, a naive enumeration of all integer h -tuples takes quasi-polynomial time. The crucial observation is that the k_i 's are not independent. They must satisfy the additional constraint that $\sum_i k_i \leq h/\epsilon^2$ since the total profit from all the sets S_1, \dots, S_h cannot exceed OPT . This constraint limits the number of tuples of relevance. We make use of the following claims.

CLAIM 2.4. *Let f be the number of g -tuples of nonnegative integers such that the sum of tuple coordinates is equal to d . Then $f = \binom{d+g-1}{g-1}$. If $d + g < \alpha g$, then $f = O(e^{\alpha g})$.*

Proof. The first part of the claim is elementary counting. If $d + g < \alpha g$, then $f \leq \binom{\alpha g}{g-1} \leq (\alpha g)^{g-1}/(g-1)!$. Using Stirling's formula we can approximate $(g-1)!$ by $\sqrt{2\pi(g-1)}((g-1)/e)^{g-1}$. Thus $f = O((e\alpha)^{g-1}) = O(e^{\alpha g})$. \square

CLAIM 2.5. *Let $h \in [4\epsilon^{-1} \ln n]$. Then the number of h -tuples (k_1, \dots, k_h) such that $k_i \in [h/\epsilon^2]$ and $\sum_i k_i \leq h/\epsilon^2$ is $O(n^{O(1/\epsilon^3)})$.*

Proof. The number of tuples satisfying the claim is easily seen to be $\binom{h/\epsilon^2+h}{h}$. We now apply the bound from Claim 2.4; we have $\alpha = (1 + 1/\epsilon^2)$ and $g = 4\epsilon^{-1} \ln n + 1$ and hence we get an upper bound of $e^{(4\epsilon^{-3} \ln n + 1 + \epsilon^{-2})}$. The claim follows. \square

Using the restricted number of distinct profit values we can also reduce the number of *distinct sizes* in the given instance to $O(\ln n)$. This property will be crucial in packing the items. The basic idea is *shifting*, an idea that is used in approximation schemes for bin packing [3]. Let A be a set of g items with identical profit but perhaps differing sizes. We order items in A in nondecreasing order of sizes and divide them into $t = (1 + 1/\epsilon)$ groups A_1, \dots, A_t with A_1, \dots, A_{t-1} containing $\lfloor g/t \rfloor$ items each and A_t containing $(g \bmod t)$ items. We discard the items in A_{t-1} , and for each $i < t - 1$ we increase the size of every item in A_i to the size of the smallest item in A_{i+1} . Since A is ordered by size, no item in A_i is larger than the smallest item in A_{i+1} for each $1 \leq i < t$. It is easy to see that if A has a feasible packing, then the modified instance also has a feasible packing. We discard at most an ϵ fraction of the profit and the modified sizes have at most $2/\epsilon$ distinct values. Applying this to each profit class we obtain an instance with $O(\epsilon^{-2} \ln n)$ distinct size values.

LEMMA 2.6. *Given an instance $I = (\mathcal{B}, \mathcal{S})$ with n items we can obtain in polynomial time $v = n^{O(1/\epsilon^3)}$ instances I_1, \dots, I_v such that*

- for $1 \leq j \leq v$, $I_j = (\mathcal{B}, \mathcal{S}_j)$;
- for $1 \leq j \leq v$, items in \mathcal{S}_j have $O(\epsilon^{-1} \ln n)$ distinct profit values;
- for $1 \leq j \leq v$, items in \mathcal{S}_j have $O(\epsilon^{-2} \ln n)$ distinct size values;
- there is an index ℓ , $1 \leq \ell \leq v$, such that \mathcal{S}_ℓ has a feasible packing in \mathcal{B} and $p(\mathcal{S}_\ell) \geq (1 - O(\epsilon))\text{OPT}(I)$.

Proof. As indicated earlier, we can guess a value \mathcal{O} such that $(1 - \epsilon)\text{OPT} \leq \mathcal{O} \leq \text{OPT}$ from $O(\epsilon^{-1} \ln n)$ values. For each guess for \mathcal{O} we round profits of items to geometric powers (see Lemma 2.2) and guess the partition of \mathcal{O} among the profit classes. The number of guesses for the partition is $n^{O(1/\epsilon^3)}$. Therefore the distinct number of instances is $n^{O(1/\epsilon^3)}$. Each instance is modified to reduce the number of distinct sizes. Each step potentially loses a $(1 - \epsilon)$ factor, so overall we lose a $(1 - O(\epsilon))$ factor in the profit. \square

We will assume for the next section that we have guessed the correct set of items and that they are partitioned into $O(\epsilon^{-2} \ln n)$ sets, with each set containing items

of the same size. We denote by U_i the items of the i th size value and by n_i the quantity $|U_i|$.

2.3. Packing items. From Lemma 2.6 we obtain a restricted set of instances in terms of item profits and sizes. We also need some structure in the bins and we start by describing the necessary transformations.

2.3.1. Structuring the bins. Assume without loss of generality that the smallest bin capacity is 1. We order the bins in increasing order of their capacity and partition them into *blocks* B_0, B_1, \dots, B_ℓ such that block B_i consists of all bins x with $(1 + \epsilon)^i \leq c(x) < (1 + \epsilon)^{i+1}$. Let m_i denote the number of bins in block B_i .

DEFINITION 2.7 (small and large blocks). *A block B_i of bins is called a small bin block if $m_i \leq 1/\epsilon$; it is called large otherwise.*

Let Q be the set of indices i such that B_i is small. Define Q' to be the set of $t = 1/\epsilon + \lceil 4\epsilon^{-1} \ln 1/\epsilon \rceil$ largest indices in the set Q . Note that we are choosing from Q the blocks with the largest indices and not the blocks with the most number of bins. Let B_Q and $B_{Q'}$ be the sets of all bins in the blocks specified by the index sets Q and Q' , respectively. The following lemma makes use of the property of geometrically increasing bin capacities.

LEMMA 2.8. *Let U be a set of items that can be packed in the bins B_Q . There exists a set $U' \subseteq U$ such that U' can be packed into the bins $B_{Q'}$, and $p(U') \geq (1 - \epsilon) \cdot p(U)$.*

Proof. Fix some packing of U in the bins B_Q . Consider the largest $1/\epsilon$ bins in B_Q . One of these bins has a profit less than $\epsilon p(U)$. Without loss of generality, assume its capacity is 1. We will remove the items packed in this bin and use it to pack items from smaller bins. Let B_i be the block containing this bin. Let j be the largest index in Q such that $j < i - 4\epsilon^{-1} \ln 1/\epsilon$. If no such j exists, $Q' = Q$ and there is nothing to prove. For any $k \leq j$, a bin in block B_k has capacity at most $1/(1 + \epsilon)^{i-k}$ since the bin from B_i had capacity 1 and the bin capacities decrease geometrically with index. Thus the bin capacity in B_k is at most $(1 + \epsilon)^{j-i+1}/(1 + \epsilon)^{j-k+1} \leq \epsilon^2/(1 + \epsilon)^{j-k+1}$. The latter inequality follows from the fact that $j - i + 1 \leq -4\epsilon^{-1} \ln 1/\epsilon$ and $(1 + \epsilon)^{-4\epsilon^{-1} \ln 1/\epsilon} \leq \epsilon^2$. Since B_k is a small bin block, it has no more than $1/\epsilon$ bins; therefore the total capacity of all bins in B_k is at most $\epsilon/(1 + \epsilon)^{j-k+1}$. Hence, the total capacity of bins in small bin blocks with indices less than or equal to j is $\sum_{k \leq j} \epsilon/(1 + \epsilon)^{j-k+1}$, which is at most 1. Therefore, we can pack all the items in blocks B_k with $k \in B_Q, k \leq j$ in the bin we picked. The total number of blocks in Q between i and j is $4\epsilon^{-1} \ln 1/\epsilon$. Each of the $1/\epsilon$ largest bins in B_Q could be in their own blocks. Hence the largest t indices from Q would contain all these blocks. From the above, we conclude that bins of blocks with indices in Q' are sufficient to pack a set $U' \subseteq U$ such that $p(U') \geq (1 - \epsilon) \cdot p(U)$. \square

Therefore we can retain the t small bin blocks from Q' and discard the blocks with indices in $Q \setminus Q'$. Hence from here on we assume that the given instance is modified to satisfy $|Q| \leq t$, and it follows that the total number of bins in small bin blocks is at most t/ϵ . When the number of bins is fixed, a PTAS is known (implicit in earlier work) even for the GAP. The basic idea in this PTAS will be useful to us in handling small bin blocks. For large bin blocks, the advantage, as we shall see later, is that we can exceed the number of bins used by an ϵ fraction. The main task is to integrate the allocation and packing of items between the different sets of bins. We do this in three steps that are outlined below.

For the rest of the section we assume that we have a set of items that can be feasibly packed in the given set of bins. We implicitly refer to some fixed feasible packing as the *optimal* solution.

2.3.2. Packing the most profitable items into small bin blocks. We guess, for each bin b in B_Q , the $1/\epsilon$ most profitable items that are packed in b in the optimal solution. The number of guesses needed is $n^{O(\ln(1/\epsilon)/\epsilon^3)}$.

2.3.3. Packing large items into large bin blocks. The second step is to select items and pack them in large bin blocks. We say that an item is packed as a *large item* if its size is at least ϵ times the capacity of the bin in which it is packed. Since the capacities of the blocks are increasing geometrically, an item can be packed as a large item in at most $f = \lceil 2\epsilon^{-1} \ln 1/\epsilon \rceil$ different blocks. Our goal is to guess all the items that are packed as large and also to which blocks they are assigned. We do this approximately as follows.

Let n_i be the number of items of the i th size class U_i , and let ℓ_i be the number packed as large in some optimal solution. Let $f_i \leq f$ be the number of blocks in which items of U_i can be packed as large. Let ℓ_i^j , $1 \leq j \leq f_i$, be the number of items packed in each of those blocks. If $\ell_i^j \leq \frac{\epsilon}{f_i} n_i$, we can discard those items, overall losing at most an ϵ fraction of the profit from U_i . Our objective is to guess a number h_i^j such that $(1 - \epsilon)\ell_i^j \leq h_i^j \leq \ell_i^j$. The number of guesses required to obtain a single h_i^j is bounded by $g = 2\epsilon^{-1} \ln f_i/\epsilon$, and therefore the total number of guesses for all h_i^j is bounded by g^f . Using f as an upper bound for f_i and simplifying we claim an upper bound of $2^{O(1/\epsilon^3)}$. Therefore the total number of guesses required for all the $O(\epsilon^{-2} \ln n)$ size classes is bounded by $n^{O(1/\epsilon^5)}$. Here is where we take advantage of the fact that the number of distinct sizes is small (logarithmic).

Suppose we have correctly assigned all large items to their respective bin blocks. We describe now a procedure for finding a feasible packing of these items. Here we ignore the potential interaction between items that are packed as large and those packed as small. We can focus on a specific block since the large items are now partitioned between the blocks. Note that even within a single block the large items could contain $\Omega(\ln n)$ distinct sizes. The abstract problem that we have is the following. Given a collection of m bins with capacities in the range $[1, 1 + \epsilon)$, and a set of n items with sizes in the range $(\epsilon, 1 + \epsilon)$, decide if there is a feasible packing for them. We do not know if this problem can be solved in polynomial time when the number of distinct sizes is $O(\ln n)$. Here we take a different approach. We obtain a relaxation by allowing use of *extra* bins to pack the items. However, we restrict the capacity of the extra bins to be 1. We give an algorithm that either decides that the instance is infeasible or gives a packing with at most an additional ϵm bins of capacity 1.

The first step in the algorithm is to pack the items of size strictly greater than 1. Let L be these set of items. Consider items of L in nondecreasing order of their sizes. When considering an item of size s , find the smallest size bin available that can accommodate it. If no such bin exists we declare that the items cannot be packed. Otherwise we pack the item into the bin and remove the bin from the available set of bins.

LEMMA 2.9. *If the algorithm fails, then there is no feasible packing for L . Further, if there is a feasible packing for all the items, then there is one that respects the packing of L produced by the above algorithm.*

Proof. In our instance, each bin's capacity is at most $1 + \epsilon$ and every item is of size strictly larger than ϵ . Therefore each item of L is packed in a bin by itself.

Suppose there are two bins x and y and an item from L of size s such that $c(x) > c(y) \geq s$. Consider any feasible packing of the items into the bins in which s is packed into x , and y does not contain any item from L . Then it is easy to see that we

can swap s to y and the items in y into x without affecting feasibility. Similarly, we can argue that if s_1 and s_2 are two items from L such that $s_1 > s_2$, then s_1 occupies a larger bin than s_2 . Using these swapping arguments we can see that the properties described in the lemma are satisfied. \square

From the above lemma, we can restrict ourselves to packing items with sizes in $(\epsilon, 1]$ into the bins that remain after packing L . Let m' be the number of bins that remain. If a feasible packing exists, the shifting technique for bin packing [3] can be adapted in a direct fashion to pack the items using $\epsilon m'$ additional bins of size 1 each. We briefly describe the algorithm. Let n' be the number of items to be packed. We observe that each bin can accommodate at most $1/\epsilon + 1$ items. Thus $m'(1/\epsilon + 1) \geq n'$. If $m' \leq 1/\epsilon$ we can check for a feasible packing by brute force enumeration. Otherwise let $t = 2/\epsilon^2$. The items are arranged in nonincreasing order of their sizes and grouped into sets $H_1, H_2, \dots, H_t, H_{t+1}$ such that $|H_1| = |H_2| = \dots = |H_t| = n'/t$ and $H_{t+1} = n' \pmod{t}$. Items in the first group H_1 that contains the largest items are each assigned to a separate bin of size 1. For $2 \leq i \leq t + 1$, the sizes of the items in H_i are uniformly set to be the size of the smallest item in H_{i-1} . It is clear that the rounded up items have a packing in the m' bins if the original items had a packing. The rounded up items have only t distinct sizes, and dynamic programming can be applied to test the feasibility of packing these items in the given m' bins in $O(n^{O(t)})$ time. Note that the number of extra bins we use is $|H_1| = n'/t = \epsilon^2 n'/2 \leq \epsilon m'$ since $m'(1/\epsilon + 1) \geq n'$. Thus we obtain the following lemma.

LEMMA 2.10. *Given $m \geq 1/\epsilon$ bins of capacities in the range $[1, 1 + \epsilon)$ and items of sizes in the range $(\epsilon, 1 + \epsilon)$, there is an $n^{O(1/\epsilon^2)}$ -time algorithm that either decides that there is no feasible packing for the items or returns a feasible packing using at most ϵm extra bins of capacity 1.*

We eliminate the extra bins later by picking the m most profitable among them and discarding the items packed in the rest. The restriction on the size and number of extra bins is motivated by the elimination procedure. In order to use extra bins the quantity ϵm needs to be at least 1. This is the reason to distinguish between small and large bin blocks. For a large bin block B_i let E_i be the *extra* bins used in packing the large items. We note that $|E_i| \leq \epsilon m' \leq \epsilon m_i$.

2.3.4. Packing the remaining items. The third and last step of the algorithm is to pack the remaining items, which we denote by R . At this stage we have a packing of the $1/\epsilon$ most profitable items in each of the bins in B_Q (bins in small bin blocks) and a feasible packing of the large items in the rest of the bins. For each bin $b_j \in \mathcal{B}$ let Y_j denote the set of items already packed into b_j in the first two steps. The item set R is packed via a linear programming (LP) approach. In particular, we use the generalized assignment formulation with the following constraints.

1. Each remaining item must be assigned to some bin.
2. An item y can be assigned to a bin b_j in a large bin block B_i only if $s(y) \leq \epsilon \cdot (1 + \epsilon)^i$. In other words, y should be small for all bins in B_i .
3. An item y can be assigned to a bin b_j in a small bin block only if $p(y) \leq \frac{\epsilon}{1+\epsilon} p(Y_j)$ and $|Y_j| \geq 1/\epsilon$.

Constraints 2 and 3 are based on the assumption that we have correctly guessed in the first two steps of the packing procedure. We make the formulation more precise now. Note that we only check for feasibility. The variable x_{ij} denotes the fraction of item i that is assigned to bin j . Let V be the set of item-bin pairs (i, j) such that i cannot be packed into b_j due to constraints 2 and 3. The precise LP formulation is

given below:

$$\begin{aligned} \sum_j x_{ij} &= 1, && \text{item } i, \\ \sum_i s(y_i) \cdot x_{ij} &\leq c(b_j), && \text{bin } j, \\ x_{ij} &= 0, && (i, j) \in V, \\ x_{ij} &= 1, && i \in Y_j, \\ x_{ij} &\geq 0, && (i, j) \in V. \end{aligned}$$

LEMMA 2.11. *The LP formulation above has a feasible solution if the guesses for the item set, the items packed as large, and those packed in small bin blocks are correct.*

Proof. Consider a feasible integral packing of the items in the bins (which by assumption exists) and let \bar{x} denote that solution. We will use \bar{x} to construct a feasible fractional solution x for the LP above. Note that \bar{x} need not satisfy the constraints imposed by V and the Y_j 's in the LP above.

Let $\text{blk}(j)$ denote the block that contains the bin b_j . We ensure the following constraint: if $\bar{x}_{ij} = 1$, then $\sum_{\{l|\text{blk}(l)=\text{blk}(j)\}} x_{il} = 1$. In other words, we fractionally assign each item to the same block that the optimal solution does. We treat the small and large bin blocks separately.

For a j where $\text{blk}(j)$ is small we set $x_{ij} = \bar{x}_{ij}$. If we had correctly guessed the largest profit items in small bin blocks, this assignment is consistent with V and Y_j .

Consider a large bin block B_k . By our assumption, we already have an integral assignment for the set of large items that \bar{x} assigns to B_k . Let S_k be the small items that are assigned by \bar{x} to B_k . We claim that S_k can be packed fractionally in B_k irrespective of the assignment of the large items. Clearly, there is enough fractional capacity. Since the sizes of the items do not change with the bins, any greedy fractional packing that does not waste capacity gives a feasible packing. \square

Let x_{ij} be a feasible fractional solution to the above formulation. Lenstra, Shmoys, and Tardos [21] and Shmoys and Tardos [28] show how a basic feasible solution to the linear program for GAP can be transformed into an integral solution that violates the capacities only slightly. We apply their transformation to x_{ij} and obtain a 0-1 solution \bar{x}_{ij} with the following properties.

1. If $x_{ij} = 0$, then $\bar{x}_{ij} = 0$, and if $x_{ij} = 1$, then $\bar{x}_{ij} = 1$.
2. For each bin b_j , either $\sum_i \bar{x}_{ij} \leq c(b_j)$ or there is an item $k(j)$ such that $\sum_{i \neq k(j)} \bar{x}_{ij} \leq c(b_j)$ and $x_{ik(j)} < 1$. We call this item $k(j)$ the *violating* item for bin b_j .

Thus we can find an integral solution where each bin's capacity is exceeded by at most one item. Further the items assigned to the bins satisfy the constraints specified by V ; that is, $\bar{x}_{ij} = 0$ if $(i, j) \in V$. The integral solution to the LP also defines an allocation of items to each block. Let P_i be the total profit associated with all items assigned to bins in block B_i . Then clearly $\mathcal{O} = \sum_{i \geq 0} P_i$. However, we have an infeasible solution since bin capacities are violated in the rounded solution \bar{x}_{ij} . We modify this solution to create a feasible solution such that in each block we obtain a profit of at least $(1 - 3\epsilon)P_i$.

Large bin blocks. Let B_i be a large bin block, and without loss of generality assume that bin capacities in B_i are in the range $[1, 1 + \epsilon)$. By constraint 2 on the

assignment, the size of any violating item in B_i is less than ϵ and there are at most m_i of them. For all $\epsilon < 1/2$ we conclude that at most $2\epsilon m_i$ extra bins of capacity 1 each are sufficient to pack all the violating items of B_i . Recall from Lemma 2.10 that we may have used ϵm_i extra bins in packing the large items as well. Thus the total number of extra bins of capacity 1, denoted by E'_i , is at most $3\epsilon m_i$. Thus all items assigned to bins in B_i have a feasible integral assignment in the set $E'_i \cup B_i$. Now clearly the m_i most profitable bins in the collection $E'_i \cup B_i$ must have a total associated profit of at least $P_i/(1 + 3\epsilon)$. Moreover, it is easy to verify that all the items in these m_i bins can be packed in the bins of B_i itself.

Small bin blocks. Consider now a small bin block B_i . By constraint 3 on the assignment, we know that the profit associated with the violating item in any bin b_j of B_i is at most $\frac{\epsilon}{(1+\epsilon)}p(Y_j)$. Thus we can simply discard all the violating items assigned to bins in B_i , and we obtain a feasible solution of profit value at least $P_i/(1 + \epsilon)$.

This gives a feasible integral solution with total profit value at least $\sum_{i \geq 0} P_i/(1 + 3\epsilon)$. Putting together the guessing and packing steps we obtain our main result.

THEOREM 2.12. *There is a PTAS for the multiple knapsack problem.*

3. Generalized assignment problem (GAP). We start by showing that even highly restricted cases of GAP are APX-hard. Then we sketch a 2-approximation algorithm for GAP that easily follows from the work of Shmoys and Tardos [28] on the Min GAP problem.

3.1. APX-hardness of restricted instances. We reduce the maximum 3-bounded 3-dimensional matching (3DM) problem [8, 16] (defined formally below) in an approximation-preserving manner to highly restricted instances of GAP.

DEFINITION 3.1 (3-bounded 3DM (3DM-3)). *We are given a set $T \subseteq X \times Y \times Z$, where $|X| = |Y| = |Z| = n$. A matching in T is a subset $M \subseteq T$ such that no elements in M agree in any coordinate. The goal is to find a matching in T of largest cardinality. A 3-bounded instance is one in which the number of occurrences of any element of $X \cup Y \cup Z$ in T is at most 3.*

Kann [16] showed that 3DM-3 is APX-hard; that is, there exists an $\epsilon_0 > 0$ such that it is NP-hard to decide whether an instance has a matching of size n or if every matching has size at most $(1 - \epsilon_0)n$. In what follows, we denote by m the number of hyperedges in the set T .

THEOREM 3.2. *GAP is APX-hard even on instances of the following form for all positive δ .*

- $p(i, j) = 1$ for all items i and bins j .
- $s(i, j) = 1$ or $s(i, j) = 1 + \delta$ for all items i and bins j .
- $c(j) = 3$ for all bins j .

Proof. Given an instance I of 3DM-3, we create an instance $I' = (\mathcal{B}, \mathcal{S})$ of GAP as follows. In I' we have m bins b_1, \dots, b_m of capacity 3 each, one for each of the edges e_1, \dots, e_m in T . For each element i of X we have an item x_i in I' and similarly y_j for $j \in Y$ and z_k for $k \in Z$. We also have an additional $2(m - n)$ items in I' , $u_1, \dots, u_{2(m-n)}$. We set all profits to be 1. It remains to set up the sizes. For each item u_h and bin b_ℓ we set $s(u_h, b_\ell) = (1 + \delta)$. For an item x_i and bin b_ℓ we set $s(x_i, b_\ell) = 1$ if $i \in e_\ell$ and $(1 + \delta)$ otherwise. The sizes of items y_j and z_k are set similarly.

We claim that 3 items can fit in a bin b_ℓ if and only if they are the elements of the edge e_ℓ . Thus bins with 3 items correspond to a matching in T . It then follows that if I has a matching of size n , then I' has a solution of value $3n + 2(m - n)$. Otherwise,

every solution to I' has value at most $3n - \epsilon_0 \cdot n + 2(m - n)$. The APX-hardness now follows from the fact that $m = O(n)$ for bounded instances. \square

A similar result can be stated if only profits are allowed to vary.

THEOREM 3.3. *GAP is APX-hard even on instances of the following form:*

- each item takes only two distinct profit values,
- each item has an identical size across all bins and there are only two distinct item sizes, and
- all bin capacities are identical.

Proof. The reduction is once again from 3DM-3. Given an instance I of 3DM-3, we create an instance $I' = (\mathcal{B}, \mathcal{S})$ of GAP as follows. In I' we have m bins b_1, \dots, b_m of capacity 3 each, one for each of the edges e_1, \dots, e_m in T . For each element i of X we have an item x_i in I' and similarly y_j for $j \in Y$ and z_k for $k \in Z$. We also have an additional $m - n$ items u_1, \dots, u_{m-n} where $s(u_h, b_\ell) = 3$ and $p(u_h, b_\ell) = 4$ for any additional item u_h and a bin b_ℓ . Fix a positive constant $\delta < 1/3$. For an item x_i and bin b_ℓ we set $p(x_i, b_\ell) = 1 + \delta$ if $i \in e_\ell$ and 1 otherwise. The profits of items y_j and z_k are set similarly. The sizes of items $x_i, y_j,$ and z_k are all set to 1 each.

It is now easy to verify that if I has a matching of size n , there exists a solution to I' of value $4(m - n) + 3n(1 + \delta)$. Otherwise, every solution to I' has value at most $4(m - n) + 3n(1 + \delta) - n\epsilon_0 \cdot \delta$. As above, the APX-hardness now follows from the fact that $m = O(n)$. \square

Notice that Theorem 3.3 is not a symmetric analogue of Theorem 3.2. In particular, we use items of two different sizes in Theorem 3.3. This is necessary as the special case of GAP where all item sizes are identical across the bins (but the profits can vary from bin to bin) is equivalent to minimum cost bipartite matching.

PROPOSITION 3.4. *There is a polynomial time algorithm to solve GAP instances where all items have identical sizes across the bins.*

3.2. A 2-approximation for GAP. Shmoys and Tardos [28] give a (1, 2) bi-criteria approximation for Min GAP. A paraphrased statement of their precise result is as follows.

THEOREM 3.5 (Shmoys and Tardos [28]). *Given a feasible instance for the cost assignment problem, there is a polynomial time algorithm that produces an integral assignment such that*

- cost of solution is no more than OPT,
- each item i assigned to a bin j satisfies $s(i, j) \leq c(j)$, and
- if a bin's capacity is violated, then there exists a single item that is assigned to the bin whose removal ensures feasibility.

We now indicate how the above theorem implies a 2-approximation for GAP. The idea is to simply convert the maximization problem to a minimization problem by turning profits into costs by setting $w(i, j) = L - p(i, j)$, where $L > \max_{i,j} p(i, j)$ is a large enough number to make all costs positive. To create a feasible instance we have an additional bin b_{m+1} of capacity 0 and for all items i we set $s(i, m + 1) = 0$ and $w(i, m + 1) = L$ (in other words $p(i, m + 1) = 0$). We then use the algorithm for cost assignment and obtain a solution with the guarantees provided in Theorem 3.5. It is easily seen that the profit obtained by the assignment is at least the optimal profit. Now we show how to obtain a feasible solution of at least half the profit. Let j be any bin whose capacity is violated by the assignment, and let i_j be the item guaranteed in Theorem 3.5. If $p(i_j, j)$ is at least half the profit of bin j , then we retain i_j and leave out the rest of the items in j . In the other case we leave out i_j . This results in a feasible solution of at least half the profit given by the LP solution. We get the following result.

PROPOSITION 3.6. *There is a 2-approximation for GAP.*

The algorithm in [28] is based on rounding an LP relaxation. For MKP an optimal solution to the linear program can be easily constructed in $O(n \log n)$ time by first sorting items by their profit to size ratio and then greedily filling them in the bins. Rounding takes $O(n^2 \log n)$ time. It is an easy observation that the integrality gap of the natural LP relaxation for GAP is 2 even on instances of MKP with identical bin capacities.

4. A greedy algorithm. We now analyze a natural greedy strategy: pack bins one at a time by applying the FPTAS for the single knapsack problem on the remaining items. Greedy(ϵ) refers to this algorithm with ϵ parameterizing the error tolerance used in the knapsack FPTAS.

CLAIM 4.1. *For instances of MKP with bins of identical capacity, the algorithm Greedy(ϵ) gives a $(\frac{e-1}{\epsilon} - O(\epsilon))$ -approximation.*

Proof. Let X be the set of items packed by some optimal solution. Let X_j denote the set of items in X that remain after Greedy packs the first $(j-1)$ bins, and let Y_j be the items packed by Greedy in the j th bin. Since the bin capacities are identical, by a simple averaging argument it is easy to see that $p(Y_j) \geq (1-\epsilon)p(X_j)/m$. Simple algebra gives the result. \square

CLAIM 4.2. *For MKP, the algorithm Greedy(ϵ) gives a $(2+\epsilon)$ -approximation.*

Proof. Let X_j denote the set of items that some fixed optimal solution assigns to the j th bin and which do not appear anywhere in Greedy's solution. Also, let Y_j denote the items that Greedy packs in the j th bin. Then we claim that $p(Y_j) \geq (1-\epsilon)p(X_j)$ since X_j was available to be packed when Greedy processed bin j . This follows from the greedy packing. Thus we obtain $\sum_{j=1}^m p(Y_j) \geq (1-\epsilon)\sum_{j=1}^m p(X_j)$. If $\sum_{j=1}^m p(X_j) \geq \text{OPT}/2$ we are done. Otherwise by definition of the X_j 's, Greedy must have packed the other half of the profit. This implies the claimed $(2+\epsilon)$ -approximation. \square

Claim 4.2 is valid even if the item sizes (but not profits) are a function of the bins, an important special case of GAP that is already APX-hard. The running time of Greedy(ϵ) is $O(mn \log 1/\epsilon + m/\epsilon^4)$ using the algorithm of Lawler [19] for the knapsack problem. Claim 4.2 has been independently observed in [2, 15].

We show an instance on which Greedy's performance is no better than 2. There are two items with sizes 1 and $\alpha < 1$ and each has a profit of 1. There are two bins with capacities 1 and α each. Greedy packs the smaller item in the big bin and obtains a profit of 1 while $\text{OPT} = 2$. This also shows that ordering bins in nonincreasing capacities does not help improve the performance of Greedy.

5. Conclusions. An interesting aspect of our guessing strategy is that it is completely independent of the number of bins and their capacities. This might prove to be useful in other variants of the knapsack problem. One application is in obtaining a PTAS for the stochastic knapsack problem with Bernoulli variables [9].

The Min GAP problem has a $(1, 2)$ bicriteria approximation, and it is NP-hard to obtain a $(1, 3/2 - \epsilon)$ -approximation. In contrast, GAP has a 2-approximation, but the known hardness of approximation is $(1 + \epsilon_0)$ for a very small but fixed ϵ_0 . Closing this gap is an interesting open problem. An $e/(e-1) + \epsilon \simeq 1.582 + \epsilon$ approximation for GAP has been obtained recently using an LP formulation [5]. Also in recent work, it has been observed that GAP is a special case of constrained submodular set function maximization, and using the results in [6], a greedy algorithm yields a $(2 + \epsilon)$ -approximation algorithm.

Another interesting problem is to obtain a PTAS for MKP with an improved running time. Though an FPTAS is ruled out even for the case of two identical bins, a PTAS with a running time of the form $f(1/\epsilon)\text{poly}(n)$ might be achievable. Such an algorithm is not known even for instances in which all bins have the same capacity.

REFERENCES

- [1] A. K. CHANDRA, D. S. HIRSCHBERG, AND C. K. WONG, *Approximate algorithms for some generalized knapsack problems*, Theoret. Comput. Sci., 3 (1976), pp. 293–304.
- [2] M. W. DAWANDE, J. R. KALAGNANAM, P. KESKINOCAK, F. S. SALMAN, AND R. RAVI, *Approximation algorithms for the multiple knapsack problem with assignment restrictions*, J. Comb. Optim., 4 (2000), pp. 171–186.
- [3] W. FERNANDEZ DE LA VEGA AND G. S. LUEKER, *Bin packing can be solved within $1 + \epsilon$ in linear time*, Combinatorica, 1 (1981), pp. 349–355.
- [4] C. E. FERREIRA, A. MARTIN, AND R. WEISMANTEL, *Solving multiple knapsack problems by cutting planes*, SIAM J. Optim., 6 (1996), pp. 858–877.
- [5] L. FLEISCHER, M. GOEMANS, V. MIRROKNI, AND M. SVIRIDENKO, *(Almost) Tight Approximation Algorithms for Maximizing General Assignment Problems*, to appear in Proceedings of the 17th Annual ACM-SIAM Symposium on Discrete Algorithms, 2005.
- [6] M. L. FISHER, G. L. NEMHAUSER, AND L. A. WOLSEY, *An analysis of approximations for maximizing submodular set functions. II*, Math. Program. Stud., 8 (1978), pp. 73–87.
- [7] A. M. FRIEZE AND M. R. B. CLARKE, *Approximation algorithms for the m -dimensional 0-1 knapsack problem: Worst-case and probabilistic analyses*, European J. Oper. Res., 15 (1984), pp. 100–109.
- [8] M. R. GAREY AND D. S. JOHNSON, *Computers and Intractability: A Guide to the Theory of NP-Completeness*, Freeman, San Francisco, CA, 1979.
- [9] A. GOEL AND P. INDYK, *Stochastic load balancing and related problems*, in Proceedings of the 40th Annual Symposium on Foundations of Computer Science, 1999, pp. 579–586.
- [10] D. S. HOCHBAUM AND D. B. SHMOYS, *A polynomial approximation scheme for scheduling on uniform processors: Using the dual approximation approach*, SIAM J. Comput., 17 (1988), pp. 539–551.
- [11] M. S. HUNG AND J. C. FISK, *An algorithm for 0-1 multiple knapsack problems*, Naval Res. Logist. Quart., 24 (1978), pp. 571–579.
- [12] M. S. HUNG AND J. C. FISK, *A heuristic routine for solving large loading problems*, Naval Res. Logist. Quart., 26 (1979), pp. 643–650.
- [13] O. H. IBARRA AND C. E. KIM, *Fast approximation algorithms for the knapsack and sum of subset problems*, J. ACM, 22 (1975), pp. 463–468.
- [14] G. INGARGIOLA AND J. F. KORSH, *An algorithm for the solution of 0-1 loading problems*, Oper. Res., 23 (1975), pp. 110–119.
- [15] J. R. KALAGNANAM, M. W. DAWANDE, M. TRUBMO, AND H. S. LEE, *Inventory Problems in Steel Industry*, Technical report RC21171, IBM T. J. Watson Research Center, 1998.
- [16] V. KANN, *Maximum bounded 3-dimensional matching is max snp-complete*, Inform. Process. Lett., 37 (1991), pp. 27–35.
- [17] N. KARMARKAR AND R. KARP, *An efficient approximation scheme for the one-dimensional bin-packing problem*, in Proceedings of the 23rd Annual Symposium on Foundations of Computer Science, 1982, pp. 312–320.
- [18] H. KELLERER, *A polynomial time approximation scheme for the multiple knapsack problem*, in Proceedings of APPROX '99, 1999, pp. 51–62.
- [19] E. L. LAWLER, *Fast approximation algorithms for knapsack problems*, Math. Oper. Res., 4 (1979), pp. 339–356.
- [20] E. L. LAWLER, J. K. LENSTRA, A. H. G. RINNOOY KAN, AND D. B. SHMOYS, *Sequencing and scheduling: Algorithms and complexity*, in Handbooks in Operations Research and Management Science, Vol. 4, S. C. Graves et al., eds., Elsevier, New York, 1993, pp. 445–522.
- [21] J. K. LENSTRA, D. B. SHMOYS, AND E. TARDOS, *Approximation algorithms for scheduling unrelated parallel machines*, Math. Program., 46 (1990), pp. 259–271.
- [22] J. LIN AND J. S. VITTER, *ϵ -approximations with minimum packing constraint*, in Proceedings of the 24th Annual ACM Symposium on Theory of Computing, 1992, pp. 771–782.
- [23] S. MARTELLO AND P. TOTH, *Solution of the zero-one multiple knapsack problem*, European J. Oper. Res., 4 (1980), pp. 322–329.

- [24] S. MARTELLO AND P. TOTH, *A bound and bound algorithm for the zero-one multiple knapsack problem*, Discrete Appl. Math., 3 (1981), pp. 275–288.
- [25] S. MARTELLO AND P. TOTH, *Heuristics algorithms for the multiple knapsack problem*, Computing, 27 (1981), pp. 93–112.
- [26] S. MARTELLO AND P. TOTH, *Knapsack Problems: Algorithms and Computer Implementations*, John Wiley and Sons, New York, 1990.
- [27] F. D. MURGOLO, *An efficient approximation scheme for variable-sized bin packing*, SIAM J. Comput., 16 (1987), pp. 149–161.
- [28] D. B. SHMOYS AND E. TARDOS, *An approximation algorithm for the generalized assignment problem*, Math. Program. A, 62 (1993), pp. 461–474.

THE RECTILINEAR STEINER ARBORESCENCE PROBLEM IS NP-COMPLETE*

WEIPING SHI[†] AND CHEN SU[‡]

Abstract. Given a set of points in the first quadrant, a rectilinear Steiner arborescence (RSA) is a directed tree rooted at the origin, containing all points, and composed solely of horizontal and vertical edges oriented from left to right, or from bottom to top. The complexity of finding an RSA with the minimum total edge length for general planar point sets has been a well-known open problem in algorithm design and VLSI routing. In this paper, we prove the problem is NP-complete in the strong sense.

Key words. Steiner tree, computational complexity, NP-complete

AMS subject classifications. 68Q17, 68U05, 90C27

DOI. 10.1137/S0097539704371353

1. Introduction. Let $P = \{p_1, p_2, \dots, p_n\}$ be a set of points in the first quadrant of E^2 , where $p_i = (x_i, y_i)$. A rectilinear Steiner arborescence (RSA) for P is a directed Steiner tree T rooted at the origin, containing all points in P , and composed solely of horizontal and vertical line segments oriented from left to right, or from bottom to top. A rectilinear Steiner minimum arborescence (RSMA) for P is an RSA for P that has the shortest possible total edge length.

The difference between an RSA and the traditional rectilinear Steiner tree is that an RSA is also a shortest distance tree with respect to the origin. Figure 1.1 shows a Steiner minimum arborescence, a Steiner minimum tree, and a minimum spanning tree for the same set of points with p_1 being the origin. For an introduction on Steiner trees, see the book by Hwang, Richards, and Winter [9].

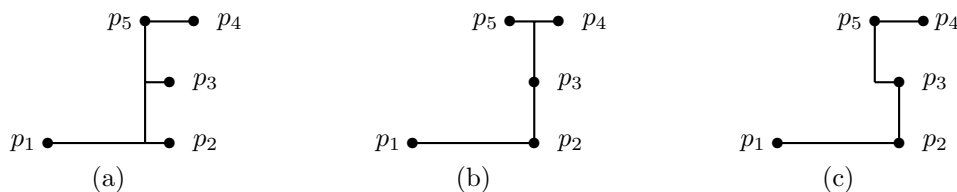


FIG. 1.1. A rectilinear Steiner arborescence (a), a rectilinear Steiner tree (b), and a rectilinear spanning tree (c).

The rectilinear Steiner arborescence problem was first studied by Nastansky, Selkow, and Stewart [13] in 1974. They proposed an integer programming formulation, which has exponential time complexity. In 1979 Laderia de Matos [10] proposed

*Received by the editors December 1, 2004; accepted for publication (in revised form) September 29, 2005; published electronically February 3, 2006. This research was supported in part by NSF grants CCR-0098329, CCR-0113668, and EIA-0223785.

<http://www.siam.org/journals/sicomp/35-3/37135.html>

[†]Department of Electrical Engineering, Texas A&M University, College Station, TX 77843 (wshi@ee.tamu.edu).

[‡]Inet Technologies, Richardson, TX 75081.

an exponential time dynamic programming algorithm. In 1985, Trubin [15] claimed the RSA problem can be solved in polynomial time. In 1992, Rao, Sadayappan, Hwang, and Shor [14] showed that Trubin's algorithm is incorrect and presented an $O(n \log n)$ time approximation algorithm that produces an RSA of length at most 2 times the optimal [14]. In 1994, Córdova and Lee [6] extended the heuristic to points in all four quadrants with the same time complexity. In 1997, Cho [3] again claimed that the RSA problem can be solved in polynomial time using a min-cost max-flow approach. Soon after, Erzin and Kahng showed Cho's claim is wrong [7]. In 2000, Lu and Ruan [12], motivated by the polynomial time approximation scheme (PTAS) of Arora [2], designed a PTAS for the RSA problem. Their PTAS runs in time $O(n^{O(c)} \log n)$ and produces an RSA of length at most $(1 + 1/c)$ times the optimal. However, whether there exists a polynomial time algorithm for the RSA problem remains open.

Because an RSMA is a shortest distance tree of minimum total length, it has important applications in VLSI routing. Cong, Leung, and Zhou [5] showed that routing trees based on RSMAs may have significantly less delay than those based on the traditional Steiner trees. Many researchers proposed efficient heuristics and exponential time exact algorithms for the RSA problem [1, 4].

2. NP-completeness.

2.1. Overall strategy. We assume that readers have the general knowledge of NP-completeness [8]. The decision version of the RSA problem is as follows:

Instance: A set of points $P = \{p_1, p_2, \dots, p_n\}$ in the plane, and a positive integer k .

Question: Is there an RSA of total edge length k or less?

The proof is a reduction from planar 3SAT, which was proven to be NP-complete in the strong sense by Lichtenstein [11]:

Instance: A set of variables $V = \{v_1, v_2, \dots, v_n\}$ and a set of clauses $C = \{c_1, c_2, \dots, c_m\}$. Each clause contains at most 3 literals. Furthermore, the bipartite graph $G = (V \cup C, E)$ is planar, where $E = \{(v_i, c_j) \mid v_i \in c_j \text{ or } \bar{v}_i \in c_j\}$.

Question: Is there an assignment for the variables so that all clauses are satisfied?

For example, Figure 2.1 is the graph of a planar 3SAT instance $c_1 = v_1 \vee v_2 \vee \bar{v}_3$ and $c_2 = \bar{v}_1 \vee \bar{v}_4$. We will use this example throughout the paper.

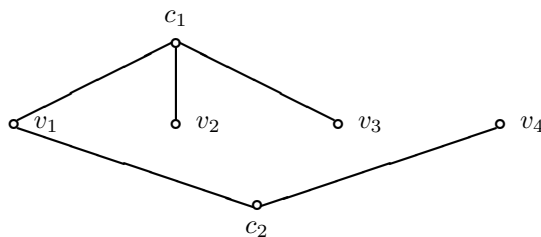


FIG. 2.1. Graph G of a planar 3SAT instance.

The reduction is through component design. For a given planar 3SAT instance, we first convert its planar graph G to a planar graph H . Then we embed H in a grid and let the embedded graph be R . Finally we replace each vertex of R by a tile. We

will use *basic tiles* to represent variables, *NOT tiles* to negate variables, *OR tiles* to compute the OR of two variables, and *clause tiles* to check if the clauses are satisfied. The length of the RSMA for the set of points so constructed will tell us whether the planar 3SAT has a satisfying solution.

2.2. Embedding. We first convert G of the given planar 3SAT instance to a planar graph H with maximum degree 3.

For each variable v_i in G , let $d(v_i)$ be the degree of v_i . Replace v_i by a path of $d(v_i)$ *variable vertices* for v_i in H : $u_{i1}, \dots, u_{id(v_i)}$ and edges $(u_{i1}, u_{i2}), \dots, (u_{id(v_i)-1}, u_{id(v_i)})$. See Figure 2.2 for an example. Each variable vertex will be connected to a clause vertex defined next.

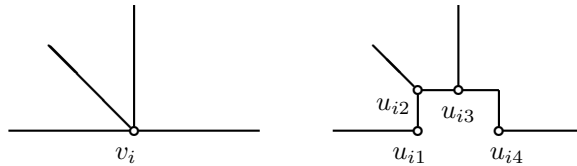


FIG. 2.2. Vertex v_i of degree 4 is replaced by a path of 4 vertices.

For each clause c_j in G that contains two variables, if $c_j = v_i \vee \overline{v_k}$, then H will contain a *clause vertex* c_j and two edges (c_j, u_i) and (c_j, u_k) , where u_i and u_k are variable vertices for v_i and v_k , respectively. Since the transformation in Figure 2.2 produced $d(v_i)$ variable vertices for each v_i , we make each edge connect to a unique variable vertex. If c_j is not in the right form, say, $c_j = \overline{v_i} \vee \overline{v_k}$, then in H we insert a *NOT vertex* w_{ij} between c_j and the variable vertex for v_i . In other words, we will have a new vertex w_{ij} and edges (c_j, w_{ij}) , (w_{ij}, u_i) , and (c_j, u_k) .

For each clause c_j in G that contains three variables, let $c_j = v_i \vee \overline{v_k} \vee \overline{v_l} = (v_i \vee \overline{v_k}) \vee \overline{v_l} = c'_j \vee \overline{v_l}$. In H , there will be an *OR vertex* c'_j that connects the variable vertices for v_i and v_k , and a *clause vertex* c_j that connects c'_j and the variable vertex for v_l . Similarly, if c_j is not in the right form, we will insert *NOT vertices* as needed.

Now H is a planar graph with maximum degree 3. Figure 2.3 shows the converted planar graph H .

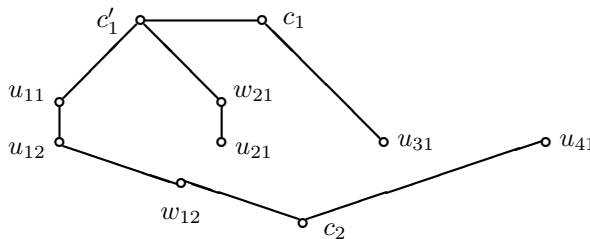


FIG. 2.3. Planar graph H converted from G .

Valiant [16] showed that any planar graph $G = (V, E)$ of maximum degree 3 has a planar embedding in a rectilinear grid of area $O(|V|^2)$. Therefore, graph H can be embedded in a rectilinear grid of area of $O((n + m)^2)$, where n and m are the number of variables and clauses. We further require the following in the grid embedding.

- (1) Vertices share an edge if and only if they are distance 1 apart.

(2) For each clause vertex $c_j = v_i \vee \overline{v_k}$, or $c_j = c'_j \vee \overline{v_k}$, the path from v_i , or c'_j , enters from the left, and the path from v_k enters from below.

(3) Each OR vertex $c'_j = v_i \vee \overline{v_k}$ occupies two horizontally adjacent vertices in the embedding, the path from v_i enters from the left, the path from v_k enters from below, and the path leading to c_j exits to the right.

These requirements can be satisfied by locally rearranging the grid embedding, increasing the size of the grid by a constant factor, and adding additional vertices and edges. For requirement (2), since v_i , c_j , and v_k is a path with no connection to other vertices in planar graph H , it is always possible to swirl the path to any direction. Figure 2.4 shows the embedding of an example clause vertex $c_j = v_i \vee \overline{v_k}$. For requirement (3), the same idea can be used. For an OR vertex $c'_j = v_i \vee \overline{v_k}$, we first swirl the path to c_j to the right. Then if v_k and v_i are in clockwise order from the path to c_j , then v_k and v_i can be swirled into proper positions. If v_k and v_i are in counterclockwise order from the path to c_j , then a crossing may occur. To avoid the crossing, note that $v_i \vee \overline{v_k} = \overline{v_k} \vee \overline{\overline{v_i}}$. Therefore we can add NOT vertices to paths for v_i and v_k and then route $\overline{v_k}$ to enter from the left and $\overline{\overline{v_i}}$ to enter from below, thereby maintaining the planarity.

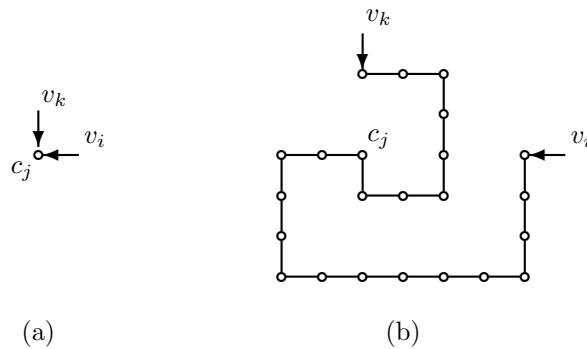


FIG. 2.4. A clause vertex with incoming paths (a). Requirement (2) is satisfied by swirling (b).

Figure 2.5 shows the grid embedding R . There are two auxiliary vertices t_1 and t_2 introduced in order to meet the above requirements.

2.3. Component design. The most important building block of our design is the *quadruped* in Figure 2.6. A quadruped consists of 4 white points q_1, \dots, q_4 , and 4 black points b_1, \dots, b_4 . The distances between the points are given in Figure 2.6, where $\alpha, \beta, \delta_1, \delta_2 > 0$. In addition, $\beta + \delta_1 > \alpha$, meaning that the Y-coordinate of q_3 is less than the Y-coordinate of q_4 , and $\alpha > \delta_1 + \delta_2$, meaning that the rectilinear distance between q_2 and q_4 is $\beta - \delta_2 + \alpha - \delta_1 > \beta$. There is no other point on or inside the region enclosed by the solid, dashed, and dotted lines, i.e., three triangles and one rectangle. We call the region the *forbidden region*.

DEFINITION 2.1. For a set of points Q that contains white points and black points, a minimum forest of Q is a set of RSAs that contains all white points in Q . Furthermore, the root of each RSA is a black point in Q , and the total edge length is minimum.

LEMMA 2.2. For a quadruped, there are only two minimum forests, shown as solid edges and dashed edges in Figure 2.6. The edge length of both minimum forests is $2(\alpha + \beta)$.

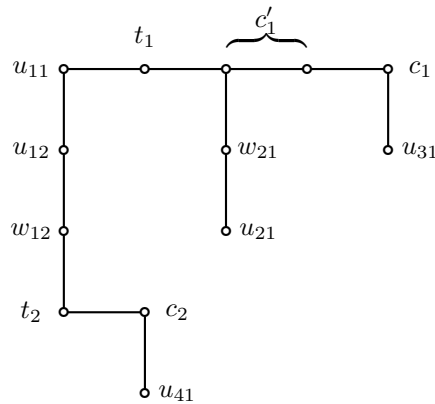


FIG. 2.5. *Embedded graph R.*

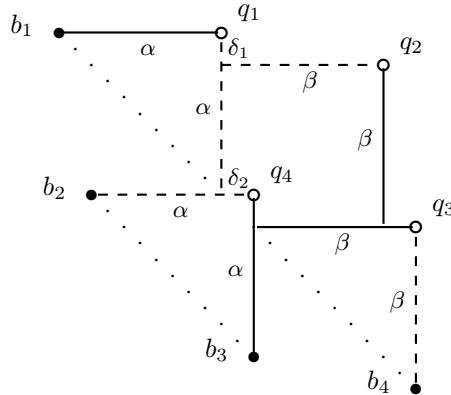


FIG. 2.6. *A quadruped and its forbidden region. The two minimum forests both have total edge length $2\alpha + 2\beta$.*

Proof. To connect q_4 , we need a path of length at least α . To connect q_1 , we also need a path of length at least α , even with the help of the path for q_4 . Similarly, to connect q_2 and q_3 , we need two paths each of length at least β . Hence, the lower bound is $2(\alpha + \beta)$.

On the other hand, to connect q_1, q_2, \dots, q_4 with the minimum edge length $2(\alpha + \beta)$, each point must use a path that equals the lower bound. Starting with q_2 , we can use either the horizontal edge or the vertical edge, both of length β . If we use the vertical edge, then we must connect q_3 with the horizontal edge, q_4 with the vertical edge, and q_1 with the horizontal edge. This gives a total length of $2(\alpha + \beta)$ shown as the solid edges. If we use the horizontal edge for q_2 , we will end up with the dashed edges, also of length $2(\alpha + \beta)$. \square

We are now ready to define the tiles. The height and width of all tiles, except for the OR tile, are both 96. The height of the OR tile is 96, and the width is 192, twice the width of other tiles.

A basic tile is made of overlapping quadrupeds as shown in Figure 2.7.

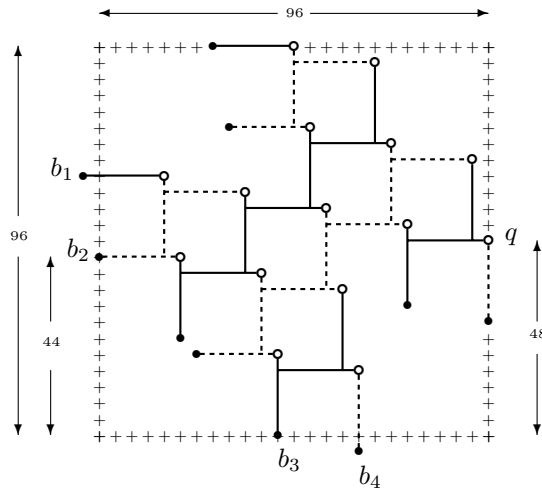


FIG. 2.7. A basic tile made of overlapping quadrupeds, where $\alpha = \beta = 20$ and $\delta_1 = \delta_2 = 4$.

LEMMA 2.3. *There are only two minimum forests for a basic tile, one shown as dashed edges and one shown as solid edges.*

Proof. Since the basic tile consists of overlapping quadrupeds, from Lemma 2.2, the minimum forest for one quadruped will force the minimum forests for the rest of the quadrupeds. \square

Define the *parity* π of a minimum forest of a basic tile to be 1 if the rightmost white point is connected by a horizontal edge, and 0 otherwise. In Figure 2.7, since the rightmost white point q is connected by a solid horizontal edge, the minimum forest that uses all solid edges has parity 1, and the minimum forest that uses all dashed edges has parity 0.

A set of overlapping quadrupeds such as a basic tile has the important property that once we choose the connection of any point to be solid or dashed, then the entire minimum forest must be solid or dashed. Therefore, if we insist on the connection being a minimum forest, then our choice of any point is propagated left, right, top, and bottom, by overlapping quadrupeds such as basic tiles.

Figure 2.8 shows how to place two horizontally adjacent basic tiles by deleting b_1 and b_2 of the right tile. To place two vertically adjacent basic tiles, we delete b_3 and b_4 of the top tile. Clearly, we have the following result.

LEMMA 2.4. *For two horizontally or vertically adjacent basic tiles, their minimum forests must have the same parity.*

Now we explain the NOT tile, which is used to change the parity. A horizontal NOT tile is shown in Figure 2.9. It is easy to check that a horizontal NOT tile can be placed between two basic tiles, according to the dimension specified in the figure. A vertical NOT tile is symmetric and can be obtained by reflecting a horizontal NOT tile with respect to line $y = x$. The parity of a minimum forest of a horizontal (vertical, respectively) NOT tile is 1 if the rightmost white point is connected by a horizontal (vertical, respectively) edge, and 0 otherwise. In Figure 2.9, since the rightmost white point q is connected by a solid vertical edge, the minimum forest that uses all solid edges has parity 0.

LEMMA 2.5. *If a horizontal NOT tile is placed between two basic tiles in a row (see Figure 2.10), then in the minimum forest, the parities of the two basic tiles must*

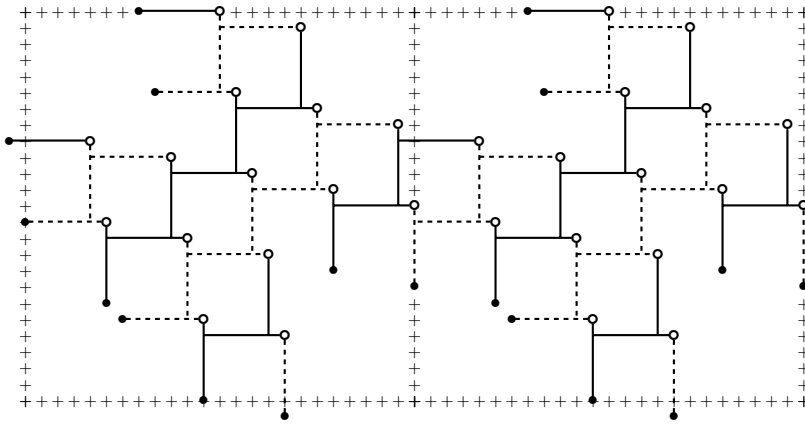


FIG. 2.8. Two adjacent basic tiles have the same parity.

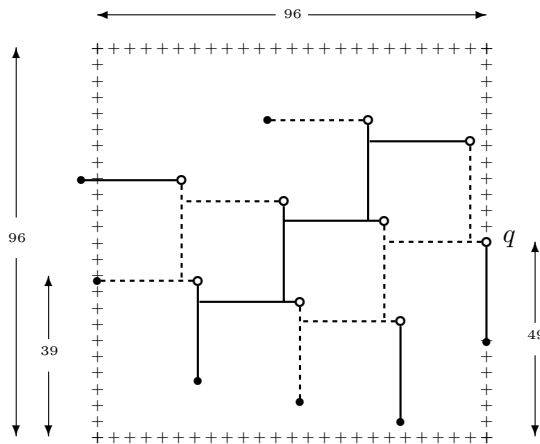


FIG. 2.9. A horizontal NOT tile, where $\alpha = \beta = 25$, $\delta_1 = 5$, and $\delta_2 = 4$.

be different. The same is true for the vertical case.

Proof. By observation, see Figures 2.9 and 2.10. It is easy to check that the interfaces between the basic tiles and the NOT tile satisfy the requirements of the quadruped. \square

A clause tile is shown in Figure 2.11. It has the same interface to the left and below as a basic tile.

LEMMA 2.6. *The length of the minimum forest of a clause tile is 108, which is achievable only if the tile to the left has parity 1, or the tile below has parity 0.*

Proof. Note that q_1, q_2, q_3 , and q_4 each requires an edge of length $\alpha = 20$. There are two ways to connect q_5 and q_6 : through edge (q_2, q_5) and edge (q_3, q_6) of length $20 + 20 = 40$ (shown as dashed edges in Figure 2.11) or through a Steiner point of length $24 + 4 = 28$ (shown as solid edges in Figure 2.11).

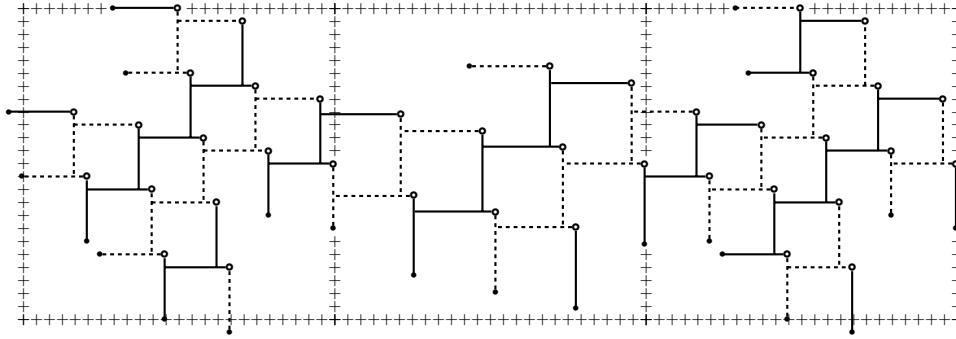


FIG. 2.10. The parity of the left basic tile is different from the parity of the right basic tile due to the NOT tile in the middle.

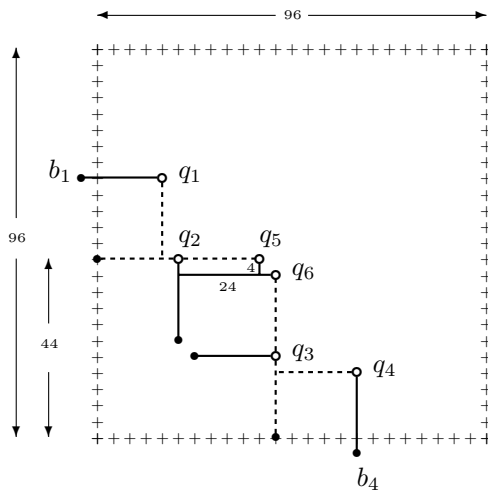


FIG. 2.11. A clause tile, where $\alpha = \beta = 20$ and $\delta_1 = \delta_2 = 4$, unless marked otherwise.

Therefore, the minimum forest has length 108, which is achievable only if q_2 or q_3 is connected by a solid edge. This in turn requires q_1 or q_4 to be connected by a solid edge. In order for q_1 to be connected by a solid edge, the parity of the forest on the left must be 1. In order for q_4 to be connected by a solid edge, the parity of the forest below must be 0. \square

Finally, we show the OR tile in Figure 2.12. There are two parts in the OR tile: The left part acts like a clause tile, and the right part adjusts the total width to 192. The parity of a minimum forest of a OR tile is 1 if the rightmost white point q is connected by a horizontal edge, and 0 otherwise.

Note that an OR tile can always have a minimum forest of parity 0. In Figure 2.12, all white points starting with q_5 and q_6 to the right can be connected by dashed edges, regardless of what happens to the left and below. However, for the OR tile to have a minimum forest of parity 1, either q_1 or q_4 must be connected by the solid edge. Since the right side of an OR tile is always a clause tile, the OR tile will try to have parity 1 if possible in order to reduce the cost of the clause tile on the right.

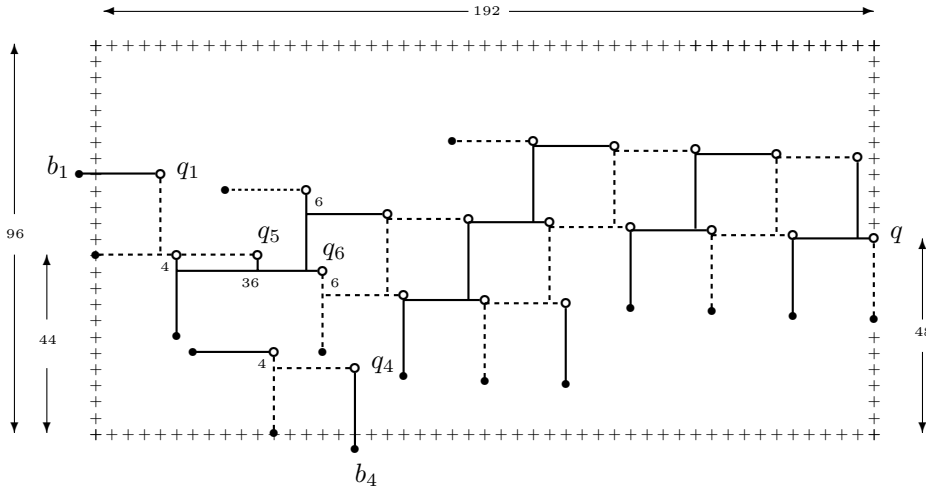


FIG. 2.12. An OR tile, where $\alpha = \beta = 20$, $\delta_1 = 1$, and $\delta_2 = 4$, unless marked otherwise.

LEMMA 2.7. *If the parity of a minimum forest of an OR tile is 1, then either the minimum forest to the left has parity 1 or the minimum forest below has parity 0. On the other hand, if either the minimum forest to the left has parity 1 or the minimum forest below has parity 0, then the parity of the OR tile can be 1.*

Proof. The proof is similar to the proof for the clause tile. In order for the OR tile to have parity 1, q_5 and q_6 cannot be connected by the dashed edges, which requires q_1 or q_4 to be connected by solid edges. Therefore, the minimum forest on the left must have parity 1 or the minimum forest below must have parity 0. \square

2.4. Main theorem. We construct an RSA instance from the 3SAT instance as follows. Each variable vertex and auxiliary vertex is replaced by a basic tile, each NOT vertex is replaced by a NOT tile, each pair of OR vertices is replaced by an OR tile, and each clause vertex is replaced by a clause tile.

To connect the black points without affecting any minimum forests for the white points, we need to add additional points. From Definition 2.1 and Lemma 2.2, if there is no point in the forbidden regions, then the properties of all tiles discussed above will not be affected. Therefore we add additional black points as follows: For each black point b_i in a basic, NOT, OR, or clause tile, we add a trail of additional black points distance 1 apart to connect b_i to the nearest black/white points to the left or below, provided that the trail does not enter any forbidden region. Such a trail always exists since any forbidden region must have a white point at the upper right corner. Figure 2.13 shows how to add trails of black points to connect some black points in a basic tile.

Since the newly added black points are not in any forbidden region, these black points will not affect the minimum forests for the quadrupeds. Furthermore, since the black points have exactly one closest neighbor to the left and below, of distance 1, all black points will be connected with a fixed edge length independent of how the white points are connected. Figure 2.14 shows the RSA instance from Figure 2.5.

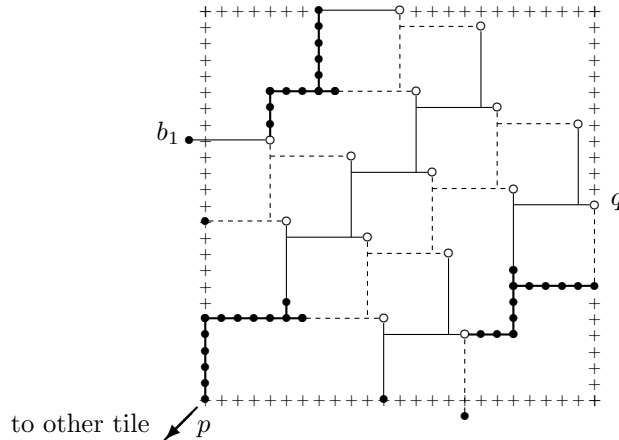


FIG. 2.13. *Trials of additional black points are added. The additional points are distance 1 apart and are not in any forbidden region. Therefore the black points will not affect how white points are connected.*

Some readers may wonder if graph G , H , or R contains a cycle, then whether the corresponding construction and connection will also contain a cycle. From Figure 2.13, it is easy to see that a cycle in G , H , or R does not correspond to a cycle in the connection. In the figure, the effect of the connection for q is propagated to b_1 , but q is not directly connected to b_1 in the Steiner tree in this tile.

THEOREM 2.8. *The RSA problem is NP-complete in the strong sense.*

Proof. It was shown in [14] that the RSA problem has the Hanan property; hence it is in NP.

For the transformed RSA instance, let L be the sum of minimum edge lengths for connecting all black points, and for the minimum forest of each basic tile, OR tile, and NOT tile. Then we claim that the set of points has an RSA of length $L + 108m$ if and only if the planar 3SAT instance has a satisfying assignment, where m is the number of clauses.

If the 3SAT is satisfiable, then for each variable v_i we make the minimum forest for the basic tiles corresponding to v_i have parity 1 if $v_i = 1$, or parity 0 if $v_i = 0$. From Lemmas 2.3 and 2.4, the parities of the basic tiles will force the parity of other tiles. Since all clauses are satisfied, from Lemmas 2.5 and 2.6, we can use the correct minimum forest for each OR tile and clause tile. Since every clause tile has edge length 108, we will have an RSA of total edge length $L + 108m$.

On the other hand, assume there is an RSA of total edge length $L + 108m$. From Lemmas 2.2, 2.3, 2.4, and 2.5, each tile must use either the dashed edges or the solid edges, the parities of adjacent tiles must match, and all clause tiles must have length 108. From Lemma 2.6, at least one variable in each clause is true. Therefore the RSA corresponds to a true assignment.

The RSA problem is strongly NP-complete because the planar 3SAT problem is strongly NP-complete, and the coordinates used in our construction are of polynomial size. \square

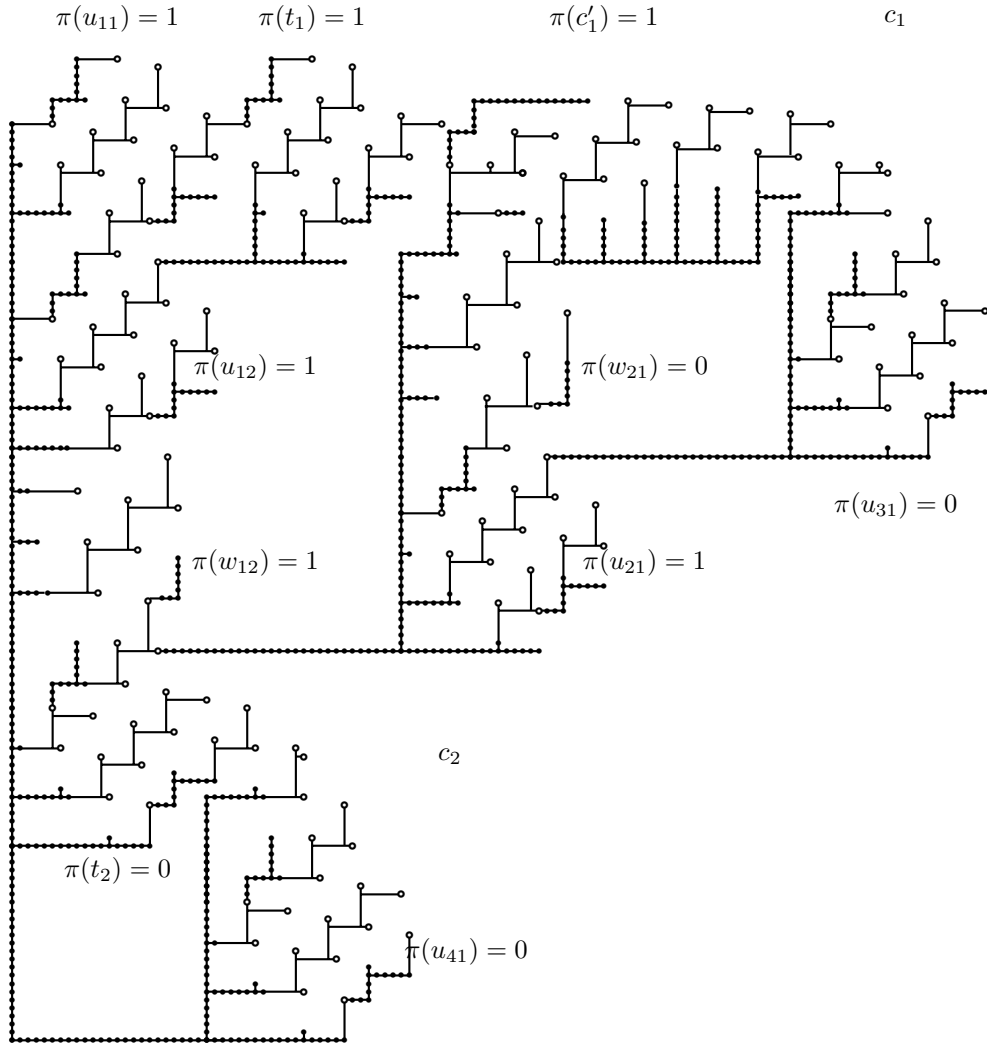


FIG. 2.14. *The transformed RSA instance, its solution, and parities of the tiles.*

Acknowledgments. The authors wish to thank Sandor Fekete and an anonymous referee for improving the presentation.

REFERENCES

- [1] M. J. ALEXANDER AND G. ROBINS, *New performance-driven FPGA routing algorithms*, IEEE Trans. Computer-Aided Design, 15 (1996), pp. 1505–1517.
- [2] S. ARORA, *Polynomial time approximation scheme for Euclidean traveling salesman and other geometric problems*, J. ACM, 45 (1998), pp. 753–782.
- [3] J. D. CHO, *Min-cost flow based min-cost rectilinear Steiner distance preserving tree construction*, in Proceedings of the International Symposium on Physical Design, ACM Press, New York, 1997, pp. 82–87.

- [4] J. CONG, A. B. KAHNG, AND K. S. LEUNG, *Efficient algorithms for the minimum shortest path Steiner arborescence problem with applications to VLSI physical design*, IEEE Trans. Computer-Aided Design, 17 (1998), pp. 24–39.
- [5] J. CONG, K. S. LEUNG, AND D. ZHOU, *Performance driven interconnect design based on distributed RC delay model*, in Proceedings of the 30th ACM/IEEE Design Automation Conference, 1993, pp. 606–611.
- [6] J. CÓRDOVA AND Y. H. LEE, *A Heuristic Algorithm for the Rectilinear Steiner Arborescence Problem*, Technical Report TR-94-025, Department of Computer Science, University of Florida, Gainesville, FL, 1994.
- [7] A. ERZIN AND A. KAHNG, *private communication*.
- [8] M. R. GAREY AND D. S. JOHNSON, *Computers and Intractability. A Guide to the Theory of NP-Completeness*, W. H. Freeman, San Francisco, 1979.
- [9] F. K. HWANG, D. S. RICHARDS, AND P. WINTER, *The Steiner Tree Problem*, North-Holland, Amsterdam, 1992.
- [10] R. R. LADERIA DE MATOS, *A Rectilinear Arborescence Problem*, Ph.D. dissertation, University of Alabama, Tuscaloosa, AL, 1979.
- [11] D. LICHTENSTEIN, *Planar formulae and their uses*, SIAM J. Comput., 11 (1982), pp. 329–343.
- [12] B. LU AND L. RUAN, *Polynomial time approximation scheme for rectilinear Steiner arborescence problem*, J. Comb. Optim., 4 (2000), pp. 357–363.
- [13] L. NASTANSKY, S. M. SELKOW, AND N. F. STEWART, *Cost-minima trees in directed acyclic graphs*, Z. Operations Res. Ser. A-B, 18 (1974), pp. 59–67.
- [14] S. K. RAO, P. SADAYAPPAN, F. K. HWANG, AND P. W. SHOR, *The rectilinear Steiner arborescence problem*, Algorithmica, 7 (1992), pp. 277–288.
- [15] V. A. TRUBIN, *Subclass of the Steiner problems on a plane with rectilinear metric*, Cybernetics, 21 (1985), pp. 320–322.
- [16] L. VALIANT, *Universality considerations in VLSI circuits*, IEEE Trans. Comput., 30 (1981), pp. 135–140.

MAPPING CYCLES AND TREES ON WRAP-AROUND BUTTERFLY GRAPHS*

MEGHANAD D. WAGH[†] AND OSMAN GUZIDE[‡]

Abstract. We give a new algebraic representation for the wrap-around butterfly interconnection network. This new representation is based on the direct product of groups and finite fields and allows an algebraic expression of the network connectivity. The abstract algebraic tools may then be employed to explore the structural properties of the butterfly. In this paper we exploit this model to map guest graphs on the butterfly. In particular, we provide designs of unit dilation mappings of all possible length cycles on butterflies. We also map the largest possible binary trees on butterfly networks with a dilation 2 if the network degree is less than 16, 3 if it is less than 32, and 4 if it is less than 64. This is a great improvement over previous results.

Key words. butterfly graphs, mathematical model, finite field, mapping, cycles, trees

AMS subject classifications. 68M07, 05C62, 68M10, 05C38

DOI. 10.1137/S0097539799365462

1. Introduction. Distributed memory parallel architectures rely upon interconnection networks to communicate data and intermediate results between processors. With the rapid advances in semiconductor technology, the computational speeds of processors have far surpassed the improvements in communication speeds. Consequently, communication between processors is threatening to become a bottleneck in parallel processing.

Improving the communication characteristics of a parallel machine is a challenging problem because of the many conflicting demands on the interconnection networks. For example, scalability and cost issues force one to have a small (and, if possible, fixed) node degree and a small number of total edges. On the other hand, performance demands a large number of processors, a small network diameter, symmetry, and the possibility of mapping of common parallel algorithm skeletons on the architecture.

The *wrap-around butterfly network* represents a good trade-off between the cost and the performance of a parallel machine. It has a large number of processors, fixed node degree, low diameter, symmetry, and ability to support a variety of parallel algorithms. A *wrap-around butterfly network* of degree $n \geq 3$, B_n , is a graph with node set $Z_n \times \{0, 1\}^n$ [7]. A node (m, V) of B_n is connected to the four nodes shown in Figure 1.1. Note that in this figure, since $m \in Z_n$, $m + 1$ and $m - 1$ are evaluated modulo n . V is an n -bit binary vector $v_{n-1}, v_{n-2}, \dots, v_0$, and 2^m refers to a length n vector with 1 in position m and 0's everywhere else. Thus an exclusive OR operation with 2^m alters exactly the m th bit of vector V . B_n is often visualized as an $n \times 2^n$ array of nodes with node (m, V) located in the m th row and V th column of the array. Each node is connected only to nodes in the neighboring rows (except for the *wrap-around* links between the nodes of the 0th and the $(n - 1)$ th rows). The edges between nodes in the same row (same m) are often called *straight edges*, and those between nodes in different rows are the *diagonal edges*.

*Received by the editors December 5, 1999; accepted for publication (in revised form) July 21, 2005; published electronically February 3, 2006.

<http://www.siam.org/journals/sicomp/35-3/36546.html>

[†]Department of Electrical and Computer Engineering, Lehigh University, Bethlehem, PA 18015 (mdw0@lehigh.edu).

[‡]Department of Computer and Information Sciences, Shepherd University, Shepherdstown, WV 25443 (oguzide@sheperd.edu).

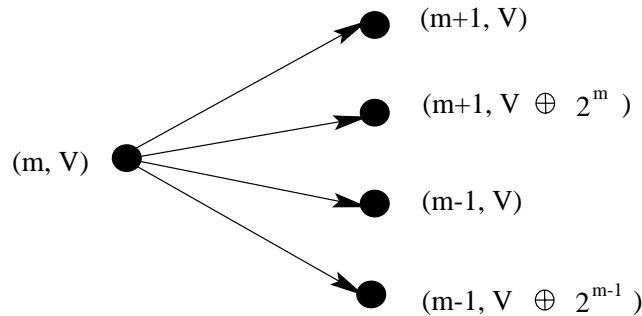


FIG. 1.1. Connections from node (m, V) in the butterfly network.

The edges in a butterfly network are bidirectional, i.e., corresponding to an edge from (m_1, V_1) to (m_2, V_2) , there is also an edge from (m_2, V_2) to (m_1, V_1) . B_n is node symmetric and has $n2^n$ nodes and $n2^{n+1}$ edges. Its node degree is 4 and its diameter is $\lfloor 3n/2 \rfloor$. The degree 4 Cayley graph, proposed recently [13], is identical to B_n [3]. Cube connected cycles are a subgraph of B_n [4]. B_n supports many parallel algorithms well [7, 5, 8, 9, 10, 12]. It is shown that one can map cycles and trees on B_n with relatively low dilation [11, 6, 2].

This paper provides a new model for the wrap-around butterfly graph using a direct product of groups and finite fields. In this model, node connectivity can be expressed as an algebraic relationship between the node labels. This allows one to explore the structural properties of the butterfly network in much more direct fashion using powerful algebraic techniques. This paper investigates the mapping of guest graphs of cycles and trees to the butterfly host graph with the help of this new model. All our mappings have unit *load*; i.e., each vertex of a guest graph is mapped to a unique butterfly node. Our mappings also have a low *dilation*; i.e., neighboring vertices of the guest graphs are mapped either to neighboring butterfly nodes (unit dilation) or on nodes between whom paths of relatively small length exists. Unit load and low dilation characterize an efficient mapping. In the case of constant node degree networks such as the wrap-around butterflies, unit load and constant dilation imply a constant congestion. Further, a unit load and unit dilation mapping is a subgraph of the host graph of butterfly.

The rest of this paper is organized as follows. In section 2, we provide the details of our new representation of B_n and prove its isomorphism to the binary node labels. Section 3 is devoted to mapping of cycles to B_n . We enumerate cycles which can *never* be subgraphs of a wrap-around butterfly graph and then provide simple procedures to design all the remaining cycle subgraphs. In particular, we show that barring a few exceptions, it is possible to map (with unit dilation) an arbitrary length cycle to B_n when n is odd, and any even length cycle when n is even. Section 4 deals with mapping trees to B_n . We show that when n is less than 16, one can map the maximal binary balanced tree to B_n with a dilation of 2. Results for larger size networks are also provided. Finally, section 5 presents our conclusions.

2. Alternate representation of the butterfly. This section presents a new model of the wrap-around butterfly using the direct product of finite groups and fields. We show that in this model, network connectivity is expressed as a simple algebraic relationship (Theorem 2.1), thereby providing powerful algebraic tools to investigate its structural properties.

In the proposed representation, nodes of B_n are labeled with the elements of $Z_n \times GF(2^n)$.¹ Thus the new node labels would be (m, X) , where $m \in Z_n$ and $X \in GF(2^n)$. Integer m and the field element X would be referred to as the first and second indices of the node, respectively. We will provide the exact equivalence between the new node labels and the ones using the binary notation later, but first we summarize important properties of finite fields used in this paper. Reader is referred to [1] for detailed description of the algebraic notions used here.

The finite field $GF(2^n)$ is an extension of $GF(2)$. Similar to $GF(2)$, it uses modulo 2 addition; i.e., for any $X \in GF(2^n)$, $X + X = 0$. Elements of $GF(2^n)$ may be enumerated as $\{0, 1, \alpha, \alpha^2, \dots, \alpha^{2^n-2}\}$, where the element α is known as the *primitive element*. $\alpha^{2^n-1} = 1$ and thus the elements of $GF(2^n)$ are closed under multiplication. The minimum degree polynomial (over $GF(2)$) of which α is a root, is called the *primitive polynomial*. Primitive polynomial has degree n and plays a central role in the design of $GF(2^n)$. Because α is a root of this degree n polynomial, elements of $GF(2^n)$ may also be expressed as polynomials (of degree at most $n - 1$) in α over $GF(2)$. One can therefore view $GF(2^n)$ as a vector space over $GF(2)$ with basis $\langle \alpha^{n-1}, \alpha^{n-2}, \dots, \alpha, 1 \rangle$.

Fields $GF(2^4)$ and $GF(2^5)$ are illustrated in Tables 2.1 and 2.2, respectively. Expressing each element of $GF(2^n)$ in basis $\langle \alpha^{n-1}, \alpha^{n-2}, \dots, \alpha, 1 \rangle$ is fairly straightforward. For example, in Table 2.1, elements 1, α , α^2 , and α^3 are already the basis elements. α^4 can be expressed using lower powers of α using the fact that α is the root of the primitive polynomial $x^4 + x + 1$. Thus $\alpha^4 + \alpha + 1 = 0$, or $\alpha^4 = \alpha + 1$. (Recall that $GF(2^n)$ uses modulo 2 additions.) The expressions for successive higher powers of α are obtained by multiplying the expressions for lower powers by α and replacing any α^4 , thus created, by $\alpha + 1$. Tables 2.1 and 2.2 are important to simplify additions between field elements. For example, using Table 2.1, one may easily add α^{10} and α^{11} in $GF(2^4)$ as $\alpha^{10} + \alpha^{11} = (\alpha^2 + \alpha + 1) + (\alpha^3 + \alpha^2 + \alpha) = \alpha^3 + \alpha = \alpha^{14}$.

Alternately, the elements of $GF(2^n)$ can be expressed over $GF(2)$ using the *dual basis* $\langle \beta_{n-1}, \beta_{n-2}, \dots, \beta_0 \rangle$. The dual basis is unique and its component β_i is defined as that element of $GF(2^n)$ which satisfies

$$(2.1) \quad \text{Tr}(\alpha^j \beta_i) = \begin{cases} 1 & \text{if } j = i, \\ 0 & \text{otherwise,} \end{cases}$$

where the *Trace* function $\text{Tr}(\cdot) : GF(2^n) \rightarrow GF(2)$ is computed as [1]

$$\text{Tr}(x) = x + x^2 + x^{2^2} + x^{2^3} + \dots + x^{2^{n-1}}.$$

$\text{Tr}(\cdot)$ is a linear function over $GF(2)$, i.e.,

$$\text{Tr}(aX + bY) = a\text{Tr}(X) + b\text{Tr}(Y), \quad a, b \in GF(2), \quad X, Y \in GF(2^n).$$

Structure of the primitive polynomial governs the relationships between the dual basis elements. For the purposes of this paper, we will need only the relationship

$$(2.2) \quad \beta_{n-1} = \alpha\beta_0.$$

¹Here, Z_n denotes the set of integers $\{0, 1, \dots, n - 1\}$ under the operation of addition modulo n and $GF(2^n)$ denotes the finite field of 2^n elements with characteristic 2.

TABLE 2.1
Structure of $GF(2^4)$.

Primitive polynomial: $x^4 + x + 1$ Elements and their relationships:	
0	$\alpha^7 = \alpha^3 + \alpha + 1$
1	$\alpha^8 = \alpha^2 + 1$
α	$\alpha^9 = \alpha^3 + \alpha$
α^2	$\alpha^{10} = \alpha^2 + \alpha + 1$
α^3	$\alpha^{11} = \alpha^3 + \alpha^2 + \alpha$
$\alpha^4 = \alpha + 1$	$\alpha^{12} = \alpha^3 + \alpha^2 + \alpha + 1$
$\alpha^5 = \alpha^2 + \alpha$	$\alpha^{13} = \alpha^3 + \alpha^2 + 1$
$\alpha^6 = \alpha^3 + \alpha^2$	$\alpha^{14} = \alpha^3 + 1$
Dual base $\langle \beta_3, \beta_2, \beta_1, \beta_0 \rangle = \langle 1, \alpha, \alpha^2, \alpha^{14} \rangle$	

TABLE 2.2
Structure of $GF(2^5)$.

Primitive polynomial: $x^5 + x^4 + x^3 + x^2 + 1$ Elements and their relationships:	
0	$\alpha^{15} = \alpha^4 + \alpha^3 + \alpha + 1$
1	$\alpha^{16} = \alpha^3 + \alpha + 1$
α	$\alpha^{17} = \alpha^4 + \alpha^2 + \alpha$
α^2	$\alpha^{18} = \alpha^4 + 1$
α^3	$\alpha^{19} = \alpha^4 + \alpha^3 + \alpha^2 + \alpha + 1$
α^4	$\alpha^{20} = \alpha + 1$
$\alpha^5 = \alpha^4 + \alpha^3 + \alpha^2 + 1$	$\alpha^{21} = \alpha^2 + \alpha$
$\alpha^6 = \alpha^2 + \alpha + 1$	$\alpha^{22} = \alpha^3 + \alpha^2$
$\alpha^7 = \alpha^3 + \alpha^2 + \alpha$	$\alpha^{23} = \alpha^4 + \alpha^3$
$\alpha^8 = \alpha^4 + \alpha^3 + \alpha^2$	$\alpha^{24} = \alpha^3 + \alpha^2 + 1$
$\alpha^9 = \alpha^2 + 1$	$\alpha^{25} = \alpha^4 + \alpha^3 + \alpha$
$\alpha^{10} = \alpha^3 + \alpha$	$\alpha^{26} = \alpha^3 + 1$
$\alpha^{11} = \alpha^4 + \alpha^2$	$\alpha^{27} = \alpha^4 + \alpha$
$\alpha^{12} = \alpha^4 + \alpha^2 + 1$	$\alpha^{28} = \alpha^4 + \alpha^3 + 1$
$\alpha^{13} = \alpha^4 + \alpha^2 + \alpha + 1$	$\alpha^{29} = \alpha^3 + \alpha^2 + \alpha + 1$
$\alpha^{14} = \alpha^4 + \alpha + 1$	$\alpha^{30} = \alpha^4 + \alpha^3 + \alpha^2 + \alpha$
Dual base $\langle \beta_4, \beta_3, \beta_2, \beta_1, \beta_0 \rangle = \langle \alpha^{20}, \alpha^9, \alpha^{26}, \alpha^{18}, \alpha^{19} \rangle$	

In order to establish the equivalence between the binary labels used in section 1 and the new labels, we use the mapping $\psi : Z_n \times \{0, 1\}^n \rightarrow Z_n \times GF(2^n)$,

$$(2.3) \quad \psi(m, v_{n-1}v_{n-2} \dots v_1v_0) = \left(m, \sum_{i=0}^{n-1} v_{(i+m) \bmod n} \beta_i \right).$$

Mapping ψ is one-to-one and onto because $\langle \beta_{n-1}, \beta_{n-2}, \dots, \beta_0 \rangle$ is a basis of $GF(2^n)$. We will show in Theorem 2.1 that ψ also preserves the connectivity of B_n .

Further, using the properties of β_i 's one can show the inverse of ψ to be

$$\psi^{-1}(m, X) = (m, v_{n-1}v_{n-2} \dots v_1v_0),$$

where

$$(2.4) \quad v_i = \text{Tr}(\alpha^{(i-m) \bmod n} X).$$

TABLE 2.3
 Equivalence between the nodes of B_5 and graph $Z_5 \times GF(2^5)$.

Label	(m, x)	Label	(m, x)	Label	(m, x)
(0, 00000)	(0, 0)	(1, 10110)	$(1, \alpha^{16})$	(3, 01100)	$(3, \alpha^8)$
(0, 00001)	$(0, \alpha^{19})$	(1, 10111)	$(1, \alpha^3)$	(3, 01101)	$(3, \alpha^{12})$
(0, 00010)	$(0, \alpha^{18})$	(1, 11000)	$(1, \alpha^{22})$	(3, 01110)	$(3, \alpha^{28})$
(0, 00011)	$(0, \alpha^7)$	(1, 11001)	$(1, \alpha^{29})$	(3, 01111)	$(3, \alpha^4)$
(0, 00100)	$(0, \alpha^{26})$	(1, 11010)	$(1, \alpha^{14})$	(3, 10000)	$(3, \alpha^{18})$
(0, 00101)	$(0, \alpha^{17})$	(1, 11011)	$(1, \alpha^4)$	(3, 10001)	$(3, \alpha^{23})$
(0, 00110)	$(0, \alpha^{23})$	(1, 11100)	$(1, \alpha^5)$	(3, 10010)	$(3, \alpha^{11})$
(0, 00111)	$(0, \alpha^6)$	(1, 11101)	$(1, \alpha^{30})$	(3, 10011)	$(3, \alpha^5)$
(0, 01000)	$(0, \alpha^9)$	(1, 11110)	$(1, \alpha)$	(3, 10100)	$(3, \alpha^{27})$
(0, 01001)	$(0, \alpha^{25})$	(1, 11111)	(1, 1)	(3, 10101)	$(3, \alpha^{15})$
(0, 01010)	$(0, \alpha^{11})$	(2, 00000)	(2, 0)	(3, 10110)	$(3, \alpha^{13})$
(0, 01011)	$(0, \alpha^{16})$	(2, 00001)	$(2, \alpha^9)$	(3, 10111)	$(3, \alpha^{30})$
(0, 01100)	$(0, \alpha^{22})$	(2, 00010)	$(2, \alpha^{20})$	(3, 11000)	$(3, \alpha^7)$
(0, 01101)	$(0, \alpha^{14})$	(2, 00011)	$(2, \alpha^{21})$	(3, 11001)	$(3, \alpha^6)$
(0, 01110)	$(0, \alpha^5)$	(2, 00100)	$(2, \alpha^{19})$	(3, 11010)	$(3, \alpha^{16})$
(0, 01111)	$(0, \alpha)$	(2, 00101)	$(2, \alpha^{25})$	(3, 11011)	$(3, \alpha)$
(0, 10000)	$(0, \alpha^{20})$	(2, 00110)	$(2, \alpha^8)$	(3, 11100)	$(3, \alpha^{24})$
(0, 10001)	$(0, \alpha^8)$	(2, 00111)	$(2, \alpha^{28})$	(3, 11101)	$(3, \alpha^2)$
(0, 10010)	$(0, \alpha^{27})$	(2, 01000)	$(2, \alpha^{18})$	(3, 11110)	$(3, \alpha^3)$
(0, 10011)	$(0, \alpha^{24})$	(2, 01001)	$(2, \alpha^{11})$	(3, 11111)	(3, 1)
(0, 10100)	$(0, \alpha^{10})$	(2, 01010)	$(2, \alpha^{27})$	(4, 00000)	(4, 0)
(0, 10101)	$(0, \alpha^{12})$	(2, 01011)	$(2, \alpha^{13})$	(4, 00001)	$(4, \alpha^{18})$
(0, 10110)	$(0, \alpha^{15})$	(2, 01100)	$(2, \alpha^7)$	(4, 00010)	$(4, \alpha^{26})$
(0, 10111)	$(0, \alpha^2)$	(2, 01101)	$(2, \alpha^{16})$	(4, 00011)	$(4, \alpha^{23})$
(0, 11000)	$(0, \alpha^{21})$	(2, 01110)	$(2, \alpha^{24})$	(4, 00100)	$(4, \alpha^9)$
(0, 11001)	$(0, \alpha^{28})$	(2, 01111)	$(2, \alpha^3)$	(4, 00101)	$(4, \alpha^{11})$
(0, 11010)	$(0, \alpha^{13})$	(2, 10000)	$(2, \alpha^{26})$	(4, 00110)	$(4, \alpha^{22})$
(0, 11011)	$(0, \alpha^3)$	(2, 10001)	$(2, \alpha^{22})$	(4, 00111)	$(4, \alpha^5)$
(0, 11100)	$(0, \alpha^{29})$	(2, 10010)	$(2, \alpha^{10})$	(4, 01000)	$(4, \alpha^{20})$
(0, 11101)	$(0, \alpha^4)$	(2, 10011)	$(2, \alpha^{29})$	(4, 01001)	$(4, \alpha^{27})$
(0, 11110)	$(0, \alpha^{30})$	(2, 10100)	$(2, \alpha^{17})$	(4, 01010)	$(4, \alpha^{10})$
(0, 11111)	(0, 1)	(2, 10101)	$(2, \alpha^{14})$	(4, 01011)	$(4, \alpha^{15})$
(1, 00000)	(1, 0)	(2, 10110)	$(2, \alpha^{12})$	(4, 01100)	$(4, \alpha^{21})$
(1, 00001)	$(1, \alpha^{20})$	(2, 10111)	$(2, \alpha^4)$	(4, 01101)	$(4, \alpha^{13})$
(1, 00010)	$(1, \alpha^{19})$	(2, 11000)	$(2, \alpha^{23})$	(4, 01110)	$(4, \alpha^{29})$
(1, 00011)	$(1, \alpha^8)$	(2, 11001)	$(2, \alpha^5)$	(4, 01111)	$(4, \alpha^{30})$
(1, 00100)	$(1, \alpha^{18})$	(2, 11010)	$(2, \alpha^{15})$	(4, 10000)	$(4, \alpha^{19})$
(1, 00101)	$(1, \alpha^{27})$	(2, 11011)	$(2, \alpha^{30})$	(4, 10001)	$(4, \alpha^7)$
(1, 00110)	$(1, \alpha^7)$	(2, 11100)	$(2, \alpha^6)$	(4, 10010)	$(4, \alpha^{17})$
(1, 00111)	$(1, \alpha^{24})$	(2, 11101)	$(2, \alpha)$	(4, 10011)	$(4, \alpha^6)$
(1, 01000)	$(1, \alpha^{26})$	(2, 11110)	$(2, \alpha^2)$	(4, 10100)	$(4, \alpha^{25})$
(1, 01001)	$(1, \alpha^{10})$	(2, 11111)	(2, 1)	(4, 10101)	$(4, \alpha^{16})$
(1, 01010)	$(1, \alpha^{17})$	(3, 00000)	(3, 0)	(4, 10110)	$(4, \alpha^{14})$
(1, 01011)	$(1, \alpha^{12})$	(3, 00001)	$(3, \alpha^{26})$	(4, 10111)	$(4, \alpha)$
(1, 01100)	$(1, \alpha^{23})$	(3, 00010)	$(3, \alpha^9)$	(4, 11000)	$(4, \alpha^8)$
(1, 01101)	$(1, \alpha^{15})$	(3, 00011)	$(3, \alpha^{22})$	(4, 11001)	$(4, \alpha^{24})$
(1, 01110)	$(1, \alpha^6)$	(3, 00100)	$(3, \alpha^{20})$	(4, 11010)	$(4, \alpha^{12})$
(1, 01111)	$(1, \alpha^2)$	(3, 00101)	$(3, \alpha^{10})$	(4, 11011)	$(4, \alpha^2)$
(1, 10000)	$(1, \alpha^9)$	(3, 00110)	$(3, \alpha^{21})$	(4, 11100)	$(4, \alpha^{28})$
(1, 10001)	$(1, \alpha^{21})$	(3, 00111)	$(3, \alpha^{29})$	(4, 11101)	$(4, \alpha^3)$
(1, 10010)	$(1, \alpha^{25})$	(3, 01000)	$(3, \alpha^{19})$	(4, 11110)	$(4, \alpha^4)$
(1, 10011)	$(1, \alpha^{28})$	(3, 01001)	$(3, \alpha^{17})$	(4, 11111)	$(4, 1)$
(1, 10100)	$(1, \alpha^{11})$	(3, 01010)	$(3, \alpha^{25})$		
(1, 10101)	$(1, \alpha^{13})$	(3, 01011)	$(3, \alpha^{14})$		

Table 2.3 provides the mapping ψ between the two representations of B_5 . In

order to illustrate the entries in this table, consider mapping of a butterfly node $(0, 01011) \in Z_n \times \{0,1\}^n$ to its new algebraic setting. The dual basis of $GF(2^5)$ given in Table 2.2 is $\langle \alpha^{20}, \alpha^9, \alpha^{26}, \alpha^{18}, \alpha^{19} \rangle$. Thus

$$\begin{aligned} \psi(0, 01011) &= (0, \alpha^9 + \alpha^{18} + \alpha^{19}) \\ &= (0, (\alpha^2 + 1) + (\alpha^4 + 1) + (\alpha^4 + \alpha^3 + \alpha^2 + \alpha + 1)) \\ &= (0, \alpha^3 + \alpha + 1) = (0, \alpha^{16}). \end{aligned}$$

Thus the butterfly node with binary label $(0, 01011)$ is renamed in the new algebraic notation as $(0, \alpha^{16})$.

We now state the central result of this section which expresses the connectivity of B_n through algebraic relationships between node labels.

THEOREM 2.1 (connectivity). *In B_n , a graph node (m, X) is connected to the four nodes $(m+1, \alpha X)$, $(m-1, \alpha^{-1}X)$, $(m+1, \alpha X + \beta_{n-1})$, and $(m-1, \alpha^{-1}X + \beta_0)$.*

Proof. Let

$$\begin{aligned} \psi(m, v_{n-1}v_{n-2} \dots v_1v_0) &= (m, X) \quad \text{and} \\ \psi(m+1, v'_{n-1}v'_{n-2} \dots v'_1v'_0) &= (m+1, \alpha X). \end{aligned}$$

Components v_i of the binary vector are related to m and X as in (2.4). Similar equation for v'_i gives

$$(2.5) \quad v'_i = \text{Tr}(\alpha^{(i-m-1) \bmod n} \alpha X).$$

Now, if $i \neq m$, then $(i-m-1) \bmod n < n-1$ and consequently $\alpha^{(i-m-1) \bmod n} \alpha = \alpha^{(i-m) \bmod n}$. On the other hand, if $i = m$, then $\alpha^{(i-m-1) \bmod n} \alpha = \alpha^n$. Using this in (2.5) gives the values of v'_i as

$$(2.6) \quad v'_i = \begin{cases} \text{Tr}(\alpha^{(i-m) \bmod n} X) & \text{if } i \neq m, \\ \text{Tr}(\alpha^n X) & \text{if } i = m. \end{cases}$$

Comparing (2.4) and (2.6) one now gets

$$(2.7) \quad v'_i = \begin{cases} v_i & \text{if } i \neq m, \\ v_m \text{ or } v_m \oplus 1 & \text{if } i = m. \end{cases}$$

The second line of (2.7) is obtained by noting that the $\text{Tr}(\alpha^n X)$ is either 0 or 1, and therefore equals either v_m or $v_m \oplus 1$. Since the binary vectors $(v_{n-1}v_{n-2} \dots v_1v_0)$ and $(v'_{n-1}v'_{n-2} \dots v'_1v'_0)$ are equal, except possibly in the m th bit, Figure 1.1 shows that nodes $(m, v_{n-1}v_{n-2} \dots v_1v_0)$ and $(m+1, v'_{n-1}v'_{n-2} \dots v'_1v'_0)$ are connected. Thus (m, X) is connected to $(m, \alpha X)$.

To show that $(m+1, \alpha X + \beta_{n-1})$ is connected to (m, X) , suppose

$$\psi(m+1, v'_{n-1}v'_{n-2} \dots v'_1v'_0) = (m+1, \alpha X + \beta_{n-1}).$$

In this case, v'_i is obtained as

$$(2.8) \quad v'_i = \text{Tr}(\alpha^{(i-m-1) \bmod n} (\alpha X + \beta_{n-1})).$$

As before, if $i \neq m$, $(i-m-1) \bmod n < n-1$. Using this and the linearity of the trace function in (2.8) gives

$$(2.9) \quad v'_i = \begin{cases} \text{Tr}(\alpha^{(i-m) \bmod n} X) + \text{Tr}(\alpha^{i-m-1 \bmod n} \beta_{n-1}) & \text{if } i \neq m, \\ \text{Tr}(\alpha^n X) + \text{Tr}(\alpha^{n-1} \beta_{n-1}) & \text{if } i = m. \end{cases}$$

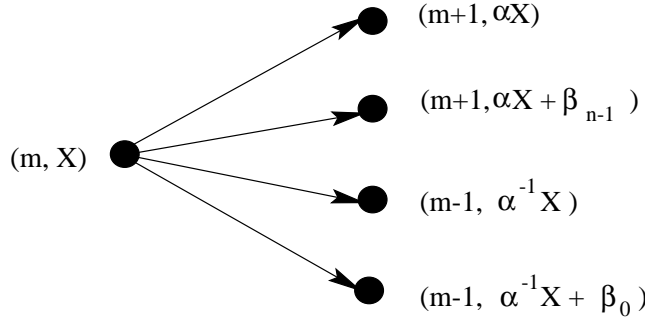


FIG. 2.1. Connections from node $(m, X) \in Z_n \times GF(2^n)$ in the butterfly network.

Employing the definition of β_{n-1} , (see (2.1)), this becomes

$$v'_i = \begin{cases} v_i & \text{if } i \neq m, \\ v_m \text{ or } v_m \oplus 1 & \text{if } i = m. \end{cases}$$

As before, $\text{Tr}(\alpha^n X) + 1$ in the second line of (2.9) is replaced by v_m or $v_m \oplus 1$ because it is either 0 or 1. Therefore from Figure 1.1, (m, X) is connected to $(m, \alpha X + \beta_{n-1})$.

The other two connections specified in the theorem can be proved similarly by substituting α^{-1} and β_0 in place of α and β_{n-1} , respectively. \square

The four edges from $(m, X) \in Z_n \times GF(2^n)$ are shown in Figure 2.1. Because of (2.2), these edges are bidirectional. It should be pointed out here that even though the four edges in Figure 2.1 map to the four edges in Figure 1.1, the exact correspondence between them is dependent upon the source node (m, X) and, in particular, on the equality of $\text{Tr}(\alpha^n X)$ and the bit v_m of binary vector V . An edge from node (m, X) to $(m+1, \alpha X)$ is sometimes a *straight edge* and sometimes a *diagonal edge*. For example, as can be seen from Table 2.3, the edge between nodes $(1, \alpha^{22})$ and $(2, \alpha^{23})$ in B_5 is a straight edge since the binary labels of these nodes are $(0, 01100)$ and $(1, 01100)$, respectively. On the other hand, the edge between nodes $(1, \alpha^{23})$ and $(2, \alpha^{24})$ is a diagonal edge since the binary labels of these nodes are $(1, 01100)$ and $(2, 01110)$, respectively. Thus the correspondence between the binary labels and their algebraic counterparts is more intricate than one might initially suppose.

Redefining B_n in the new algebraic notation allows use of simple but powerful algebraic techniques to study its structure. Also, unlike the binary representation (Figure 1.1), in the new algebraic notation (Figure 2.1) the two indices of a node change independently between connected nodes. This independence further simplifies our investigation.

3. Mapping cycles on the butterfly. This section provides comprehensive results about cycles as subgraphs of B_n . We first prove exactly which cycles are *not* subgraphs of B_n (Theorem 3.1). Then we provide simple procedures to map to B_n all the permissible cycles, i.e., those that are not enumerated in Theorem 3.1 (Theorems 3.3, 3.6, and 3.7). Earlier, Rosenberg [11] has given mappings of cycles of lengths $L = n$ or $L = n2^n - (n - 2)c$, $1 \leq k \leq n$, $0 \leq c \leq 2^k$, on B_n . His results map at most $n + 2^{n+1} - 1$ cycles of different lengths on B_n when n is even and exactly $n + 2^{n+1} - 1$ cycles when n is odd. On the other hand, we give constructions of all the cycle subgraphs of B_n that are ever possible. Thus our methods map as many as

$n2^n - (n + 5)/2$ cycles of different lengths on B_n when n is odd and at least $n2^{n-1} - 3$, when n is even.

We begin by specifying which cycles can *never* be subgraphs of B_n .

THEOREM 3.1 (impossible cycles). *Simple cycles (i.e., cycles with distinct nodes) of the following lengths L are never subgraphs of B_n :*

- (a) *odd L when n is even,*
- (b) *odd L less than n ,*
- (c) *$L = 6$ when $n = 5$ or $n \geq 7$,*
- (d) *$L = 10$ when $n = 7, n = 9, \text{ or } n \geq 11$.*

Proof. (a). From Figure 2.1, if n is even and (m, X) is connected to (m', X') , then exactly one of m and m' is odd and the other is even. This implies that for even n , B_n is a *bipartite graph*, and therefore an odd-length cycle cannot be its subgraph.

(b). Note that because of the connectivity described in Figure 2.1, the first indices of all the nodes in the cycle may be translated by the same amount to get another equivalent cycle. Thus when $L < n$, the solution can be embedded on a butterfly *without* using any wrap-around edges. But an unwrapped butterfly is a bipartite graph and therefore cannot have an odd-length cycle subgraph.

(c) and (d). These cases may be proved by enumerating all possibilities of mapping the cycle and then illustrating contradictions in each case. First note that it is impossible to have the first indices of any five consecutive nodes in a cycle to be $m, m + 1, m, m + 1$, and m . Because if $(m, X) \rightarrow (m + 1, X_1) \rightarrow (m, X_2) \rightarrow (m + 1, X_3) \rightarrow (m, X_4)$ are the five connected nodes, then from Figure 2.1, $X_1 = \alpha X + c\beta_{n-1}$, $c \in \{0, 1\}$. This gives $X_2 = X + \beta_0$. Clearly, the only choices for X_3 are $\alpha X + \beta_{n-1}$ and αX , giving X_4 equal to $X + \beta_0$ or X . Thus node (m, X_4) is not distinct and the assumed chain of five nodes does not exist.

We can now demonstrate the impossibility of cycle mapping for $L = 6$. The case of $L = 10$ can be proved similarly. Let, if possible,

$$(m_0, X_0) \rightarrow (m_1, X_1) \rightarrow (m_2, X_2) \rightarrow (m_3, X_3) \rightarrow (m_4, X_4) \rightarrow (m_5, X_5) \rightarrow (m_0, X_0)$$

denote the length 6 cycle which is a subgraph of B_n , $n \geq 7$. Clearly, the first index of all cycle nodes can be increased or decreased by the same amount, or the direction of the cycle traversal may be reversed without disturbing the connectivity. Therefore, without loss of generality, one may choose $m_0 = 0$ and $m_1 = 1$. Clearly $m_5 = 1$ as well, because one cannot go from $m_1 = 1$ to $m_5 = n - 1$ in only 4 hops since $n \geq 7$. Indices m_2, m_3 , and m_4 should satisfy two conditions: (1) cyclically successive values in the sequence $(m_0, m_1, m_2, m_3, m_4, m_5)$ change only by 1; (2) no five cyclically consecutive values in the sequence are $(m, m + 1, m, m + 1, m)$. It can be verified that under these conditions, the only possible set of values for m_0 through m_5 are $(0, 1, 2, 3, 2, 1)$. Without loss of generality, let $X_0 = X$, $X_1 = \alpha X$, and $X_5 = \alpha X + \beta_{n-1}$. Then for $c_i \in \{0, 1\}$, following successive links, one gets $X_2 = \alpha^2 X + c_1\beta_{n-1}$, $X_3 = \alpha^3 X + c_1\alpha\beta_{n-1} + c_2\beta_{n-1}$, $X_4 = \alpha^2 X + c_1\beta_{n-1} + c_2\alpha^{-1}\beta_{n-1} + c_3\beta_0$, and $X_5 = \alpha X + c_1\alpha^{-1}\beta_{n-1} + c_2\alpha^{-2}\beta_{n-1} + c_3\alpha^{-1}\beta_0 + c_4\beta_0$. Equating the two values of X_5 and then using $\beta_{n-1} = \alpha\beta_0$ give

$$\alpha^2 + (c_1 + c_4)\alpha + (c_2 + c_3) = 0.$$

But this is impossible since α cannot satisfy an equation of degree smaller than n .

When $L = 6$ and $n = 5$, the only possible set of values of m_0 through m_5 that need to be considered are $(0, 1, 2, 3, 2, 1)$, $(0, 1, 0, 4, 3, 4)$, $(0, 1, 2, 1, 0, 4)$, and $(0, 4, 3, 2, 3, 4)$. Note that if all the indices in any set are increased by a constant amount, then the

set transforms into a rotated version of the first set. For example, by adding 2 to each index of the second set, one gets set (2, 3, 2, 1, 0, 1), which is simply the first set rotated left twice. Thus by dealing with only the first set, no generality is lost. But we demonstrated earlier that this first set does not produce a valid cycle of length 6. Therefore it is impossible to have a length 6 cycle as a subgraph of B_n when $n \geq 7$ or $n = 5$. \square

To obtain cycle mappings for lengths not specified in Theorem 3.1 we proceed as follows. Theorem 3.3 gives the mappings when the cycle length L is divisible by n .² This also includes the Hamiltonian cycle. For other lengths that may be expressed as $L = Kn + 2t \leq n2^n$, for some $K > 0$ and $0 \leq t < n$, Theorem 3.6 shows that one can first design a cycle of length Kn and then attach t pairs of new nodes to it. Finally, an alternate procedure to map cycles of lengths less than $4n$ (except 6 and 10) is provided in Theorem 3.7.

In order to prove the existence of cycles in butterfly networks, we need the following lemma.

LEMMA 3.2. $n \nmid (2^n - 1)$ for any integer $n > 1$.

Proof. The lemma is obvious when n is an even integer. Further, when n is an odd prime, according to *Fermat's little theorem*, $2^n = 2 \pmod n$ which shows that for prime n , $n \nmid (2^n - 1)$. If possible, let n be the smallest odd integer such that $n \mid (2^n - 1)$. Clearly n must be composite. Let p denote the largest odd prime in n , i.e., $n = p^t p_1^{t_1} p_2^{t_2} \dots$, where p, p_1, p_2, \dots are distinct primes, $p > p_1, p_2, \dots$. Consider the group G of integers less than n and relatively prime to n under the operation of multiplication modulo n . The Euler phi-function $\phi(n)$ which represents the number of elements in G is given by

$$(3.1) \quad \phi(n) = (p - 1)p^{t-1}(p_1 - 1)p_1^{t_1-1}(p_2 - 1)p_2^{t_2-1} \dots$$

As $2 \in G$, it satisfies

$$(3.2) \quad 2^{\phi(n)} = 1 \pmod n.$$

Now, if $n \mid (2^n - 1)$, then

$$(3.3) \quad 2^n = 1 \pmod n.$$

Equations (3.2) and (3.3) imply that

$$(3.4) \quad 2^{\gcd(\phi(n), n)} = 1 \pmod n.$$

Since p is the largest prime in n , the power of p in $\phi(n)$ according to (3.1) is $t - 1$. Therefore

$$(3.5) \quad \gcd(\phi(n), n) \mid (n/p).$$

From (3.4) and (3.5) one gets

$$2^{(n/p)} = 1 \pmod n,$$

and consequently,

$$(3.6) \quad 2^{(n/p)} = 1 \pmod{(n/p)}.$$

²We use the notation $n|L$ to indicate that L is a multiple of n , and $n \nmid L$ to indicate that L is not a multiple of n .

But this is contradictory to the assumption that n is the smallest integer satisfying $n \mid (2^n - 1)$. Hence there is no such n . \square

We can now state the theorems on cycles in butterfly networks.

THEOREM 3.3 (cycles of length divisible by n). *Suppose L is an arbitrary multiple of n and $L \leq n2^n$. Then cycle of length L can be mapped to B_n with dilation 1.*

Proof. Let $g = \gcd(n, 2^n - 1)$. From Lemma 3.2, one gets $n/g \geq 2$. We consider the following two cases based on the magnitude of L .

Case 1. $L \leq n(2^n - 1)/g$. If $L = n(2^n - 1)/g$, choose any nonzero $X \in GF(2^n)$. Otherwise, $\alpha^L \neq 1$ in $GF(2^n)$; choose X as

$$(3.7) \quad X = \beta_{n-1}(1 + \alpha^L)^{-1}.$$

The required cycle may then be constructed as

$$(3.8) \quad (0, X) \rightarrow (1, \alpha X) \rightarrow (2, \alpha^2 X) \rightarrow \dots \rightarrow ((L - 1) \bmod n, \alpha^{L-1} X) \rightarrow (0, X).$$

From the graph connectivity described earlier, each node on this cycle is connected to the next. Observe that (3.7) implies that $\alpha^L X = X + \beta_{n-1}$, so the last edge in (3.8) also is valid. Further, the first component of the node label repeats with periodicity of n and the second, with periodicity $(2^n - 1)$. Therefore the same label will repeat only with a periodicity of $n(2^n - 1)/g$. Thus for $L \leq n(2^n - 1)/g$, all the nodes in the cycle are distinct.

Case 2. $L > n(2^n - 1)/g$. Partition L as

$$(3.9) \quad L = n + L_1 + L_2 + \dots + L_t, \quad \text{where } (2^n - 1) \leq L_i \leq n(2^n - 1)/g \text{ and } n \mid L_i.$$

One way to achieve this partition is to choose

$$(3.10) \quad t = \left\lceil \frac{g(L - n)}{n(2^n - 1)} \right\rceil,$$

set $L_i = n(2^n - 1)/g$ for $1 \leq i < t$, and adjust L_t to make up the total to L . If this $L_t < 2^n - 1$, then reduce L_{t-1} by some amount and increase L_t by the same amount. Since $n/g \geq 2$, $L_{t-1} \geq 2(2^n - 1)$. Therefore, one can always find an appropriate amount to shift from L_{t-1} to L_t so as to make both $L_{t-1}, L_t \geq 2^n - 1$.

To build the required cycle, first obtain t disjoint cycles C_i of lengths L_i as in Case 1. Cycles of length $n(2^n - 1)/g$ may be constructed by starting from arbitrary nonzero nodes not used in previous cycles and always going from (m, x) to $(m+1, \alpha x)$. To create cycles of length less than $n(2^n - 1)/g$, compute the second index X of the starting node according to (3.7). Use a first index such that the node has not appeared in previous cycles. Note that this is possible because a node with the same second index repeats in a cycle only with a period of $2^n - 1$. In each cycle of length less than or equal to $n(2^n - 1)/g$, such labels occur at most n/g times. Since the number of cycles, t , is at most g (see (3.10)), unused labels (m, X) will be available to start new cycles.

Finally, build a cycle C_0 of length n as

$$(0, 0) \rightarrow (1, 0) \rightarrow (2, 0) \rightarrow \dots \rightarrow (n - 1, 0) \rightarrow (0, 0).$$

It is easy to see that the neighboring nodes in C_0 are connected and are distinct from those in the previous t cycles.

These $t + 1$ cycles can be merged together to form a single cycle as follows. Since each $L_i \geq 2^n - 1$, each cycle C_i , $1 \leq i \leq t$, contains consecutive elements

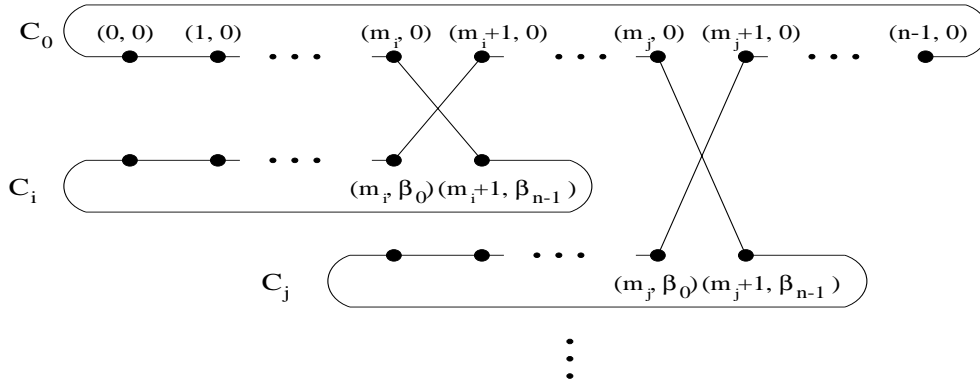


FIG. 3.1. Merging C_0 and C_i 's into a single cycle.

(m_i, β_0) and $(m_i + 1, \beta_{n-1})$ for some $0 \leq m_i < n$. To combine C_i and C_0 , add edges $(m_i, \beta_0) \rightarrow (m_i + 1, 0)$ and $(m_i + 1, \beta_{n-1}) \rightarrow (m_i, 0)$ and remove edges $(m_i, \beta_0) \rightarrow (m_i + 1, \beta_{n-1})$ and $(m_i, 0) \rightarrow (m_i + 1, 0)$. Since the m_i for each C_i is different, each C_i can be joined with C_0 at a different place. This process of cycle merging is sketched in Figure 3.1. The resultant cycle thus has the desired length L . \square

Note that for most values of n , we have $g = 1$. In fact, for $n \leq 20$, the only values of n for which $g > 1$ are 6, 12, and 18. Case 2 of Theorem 3.3 is mostly useful for these n 's. Using techniques similar to those of Theorem 3.3, it is also possible to map many disjoint cycles to B_n simultaneously. This is shown in the following corollary.

COROLLARY 3.4 (multiple cycles). *If $n|L$ and $L \leq t(2^n - 1)$ for some t , $0 < t < (n/g)$, one can map $g\lfloor n/(gt) \rfloor$ disjoint cycles of length L to B_n with dilation 1.*

Proof. Obtain X from (3.7) corresponding to the given L . Let h represent $(2^n - 1) \bmod n$. Begin the required cycles from

$$(3.11) \quad (hti_1 + i_2, X), \quad 0 \leq i_1 < \lfloor n/(tg) \rfloor, \quad 0 \leq i_2 < g,$$

and use the edges $(m, x) \rightarrow (m + 1, \alpha x)$ repeatedly until each cycle is complete.

From Theorem 3.3, it is clear that length of each cycle is L . We need only to prove that the nodes used in each cycle are distinct. Assume that, if possible, the i th node of the cycle beginning at $(hti_1 + i_2, X)$ is the same as the i' th node of the cycle beginning at $(hti'_1 + i'_2, X)$, i.e.,

$$(3.12) \quad (hti_1 + i_2 + i, \alpha^i X) = (hti'_1 + i'_2 + i', \alpha^{i'} X).$$

We will show that this implies that the two starting nodes are identical, i.e., $i_1 = i'_1$ and $i_2 = i'_2$. This being contradictory to the construction described above, we conclude that the cycles are disjoint.

By comparing the second indices of the nodes in (3.12) we get

$$(3.13) \quad i - i' = q(2^n - 1) \quad \text{for some integer } q.$$

Note that since $0 \leq i, i' < L = t(2^n - 1)$, one has $q < t$. Comparison of the first indices in (3.12) yields

$$ht(i_1 - i'_1) + (i_2 - i'_2) + q(2^n - 1) \equiv 0 \pmod{n}$$

or

$$(3.14) \quad h(t(i_1 - i'_1) + q) + (i_2 - i'_2) \equiv 0 \pmod{n}.$$

Note now from the definition of h that

$$(3.15) \quad g = \gcd(2^n - 1, n) = \gcd((2^n - 1) \bmod n, n) = \gcd(h, n).$$

By reducing each term in (3.14) modulo g (a factor of n and h), one gets

$$i_2 \equiv i'_2 \pmod{g}.$$

But since each $i_2, i'_2 < g$,

$$(3.16) \quad i_2 = i'_2.$$

Combining this with (3.14) and using (3.15) give

$$(3.17) \quad i_1 t + q \equiv i'_1 t \pmod{(n/g)}.$$

However, because of the bounds on i_1, i'_1, t , and q , one can verify that $i_1 t + q < (n/g)$ as well as $i'_1 t < (n/g)$. Therefore,

$$i_1 t + q = i'_1 t.$$

Since $q < t$, this gives

$$i_1 = i'_1. \quad \square$$

Corollary 3.4 allows one to efficiently utilize the butterfly architectures for concurrent computation of multiple algorithms, each having a cyclic communication structure. Thus, for example, in the case of B_6 , one can have 6 disjoint cycles of any length (divisible by 6) up to 60, or 3 disjoint cycles of any length (divisible by 6) up to 126.

We illustrate the construction by mapping four length 12 cycles to B_4 . To do this, we compute X from (3.7) in field $GF(2^4)$ as (refer to Table 2.1)

$$X = 1 \cdot (1 + \alpha^{12})^{-1} = \alpha^4.$$

The four disjoint cycles are then directly given by

$$\begin{aligned} &(0, \alpha^4) \rightarrow (1, \alpha^5) \rightarrow (2, \alpha^6) \rightarrow (3, \alpha^7) \rightarrow (0, \alpha^8) \rightarrow (1, \alpha^9) \rightarrow (2, \alpha^{10}) \\ &\quad \rightarrow (3, \alpha^{11}) \rightarrow (0, \alpha^{12}) \rightarrow (1, \alpha^{13}) \rightarrow (2, \alpha^{14}) \rightarrow (3, 1) \rightarrow (0, \alpha^4). \\ &(3, \alpha^4) \rightarrow (0, \alpha^5) \rightarrow (1, \alpha^6) \rightarrow (2, \alpha^7) \rightarrow (3, \alpha^8) \rightarrow (0, \alpha^9) \rightarrow (1, \alpha^{10}) \\ &\quad \rightarrow (2, \alpha^{11}) \rightarrow (3, \alpha^{12}) \rightarrow (0, \alpha^{13}) \rightarrow (1, \alpha^{14}) \rightarrow (2, 1) \rightarrow (3, \alpha^4). \\ &(2, \alpha^4) \rightarrow (3, \alpha^5) \rightarrow (0, \alpha^6) \rightarrow (1, \alpha^7) \rightarrow (2, \alpha^8) \rightarrow (3, \alpha^9) \rightarrow (0, \alpha^{10}) \\ &\quad \rightarrow (1, \alpha^{11}) \rightarrow (2, \alpha^{12}) \rightarrow (3, \alpha^{13}) \rightarrow (0, \alpha^{14}) \rightarrow (1, 1) \rightarrow (2, \alpha^4). \\ &(1, \alpha^4) \rightarrow (2, \alpha^5) \rightarrow (3, \alpha^6) \rightarrow (0, \alpha^7) \rightarrow (1, \alpha^8) \rightarrow (2, \alpha^9) \rightarrow (3, \alpha^{10}) \\ &\quad \rightarrow (0, \alpha^{11}) \rightarrow (1, \alpha^{12}) \rightarrow (2, \alpha^{13}) \rightarrow (3, \alpha^{14}) \rightarrow (0, 1) \rightarrow (1, \alpha^4). \end{aligned}$$

There is also another simple configuration of multiple cycles on B_n when $g = 1$. A cycle of length $L < n(2^n - 1)$, $n|L$, is given by

$$(0, X) \rightarrow (1, \alpha X) \rightarrow (2, \alpha^2 X) \rightarrow \dots \rightarrow ((L - 1), \alpha^{L-1} X) \rightarrow (0, X),$$

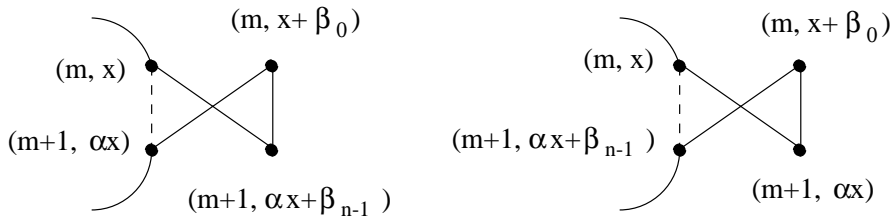


FIG. 3.2. Two cases of adding a pair of outside nodes to a cycle.

where

$$X = \beta_{n-1}(1 + \alpha^L)^{-1}.$$

It is easy to verify that all but n of the remaining nodes in B_n are also linked as a cycle. This *complementary cycle* of length $n2^n - n - L$ is given by

$$\begin{aligned} (L, \alpha^L X) &\rightarrow (L + 1, \alpha^{L+1} X) \rightarrow (L + 2, \alpha^{L+2} X) \rightarrow \dots \\ &\rightarrow (n(2^n - 1) - 1, \alpha^{n(2^n - 1) - 1} X) \rightarrow (L, \alpha^L X). \end{aligned}$$

One should also note that the only nodes which are not part of either the cycle of length L or its complementary cycle form a third cycle C_0 described earlier in Figure 3.1. Thus, when $g = 1$, the nodes of B_n can be partitioned into three cycles of lengths n , L , and $n2^n - L - n$ with the only condition on L being that it should be a multiple of n .

When $g > 1$ and cycle length $L \leq (2^n - 1)n/g$ is a multiple of n , one can similarly show that the nodes of B_n may be partitioned into g cycles of length L , g cycles of length $(2^n - 1)n/g - L$, and one cycle of length n .

We now present the result about mapping cycles of lengths that are *not* multiples of n . Our methodology is rather simple. We first form a cycle of a smaller length which is a multiple of n . Then we attach appropriately chosen pairs of outside nodes to this cycle. This process is illustrated in Figure 3.2. As shown in the figure, if the cycle link shown by the dashed line is removed and three new links are added, then the outside pair of nodes can be incorporated in the cycle. We will refer to this process as *attaching a node pair at (m, x)* . Further, the pair of nodes, (m, x) and $(m, x + \beta_0)$, which plays a crucial role in this process, will be called the pair of *companion nodes*. Note that the companion node labels have the same first index, and their second indices differ by β_0 .

For this method to succeed, it is necessary to find enough nodes in the cycle with their companion nodes outside the cycle. Lemma 3.5, to be presented later, will help us count (or in some cases, bound) the number of companion node pairs. This lemma will then be used to prove Theorem 3.6, which guarantees that there are, indeed, the required number of companion node pairs to construct cycles of any length.

For any $(m, X) \in Z_n \times GF(2^n)$, $X \neq 0$, define a *chain* starting from (m, X) to mean a set of distinct nodes

$$\{(m, X), (m + 1, \alpha X), (m + 2, \alpha^2 X), \dots, (m + T - 1, \alpha^{T-1} X)\}.$$

The number of nodes in the chain, T , will be called the length of the chain. The maximum value of T is $(2^n - 1)n/g$. Butterfly connectivity described in Figure 2.1 shows that the consecutive nodes in a chain are connected.

Let (m, α^i) be any node of B_n . We refer to the quantity $(i - m) \bmod g$ as the *partition index* of that node. Clearly, all the nodes (m_i, α^i) of a chain have the same partition index. This is because from one node to the next, m_i increases by 1 mod n and the power of α , by 1 mod $(2^n - 1)$. The $n(2^n - 1)$ nodes of B_n with nonzero second index may be separated into g partitions based on their partition index. Each partition has exactly $n(2^n - 1)/g$ nodes. Each chain is confined to a single partition. Chains of length $n(2^n - 1)/g$ occupy a complete partition.

In the light of this new terminology, one can see that when $g = 1$, a cycle formed as in (3.8) is also a chain. Further, for $g \neq 1$, cycles C_1, C_2, \dots, C_t described in Theorem 3.3 can also be viewed as chains in distinct partitions. Therefore, the number of companion node pairs that may be used to extend the cycle length can be obtained by studying companion node pairs in relation to chains. We call a companion node pair to be *within a chain* if both its nodes are in the same chain. If the two nodes are in different chains, we call that companion node pair to be *across chains*. The following lemma explores the bounds on these numbers.

LEMMA 3.5 (companion node pairs within and across chains).

(a) *The number of companion node pairs, $\Gamma(T)$, confined to a chain of length T satisfies*

$$(3.18) \quad \Gamma(T) \leq \begin{cases} \lfloor (T - 1)/n \rfloor & \text{if } T \leq 2^n - 1, \\ \lfloor (T - 1)/n \rfloor + \Gamma(T - (2^n - 1)) & \text{otherwise.} \end{cases}$$

(b) *The number of companion node pairs across two disjoint chains of lengths $T_1, T_2 \leq 2^n - 1$ is at most $\lfloor (T_1 + T_2)/n \rfloor + 1$.*

(c) *The number of companion pairs across the two chains (not necessarily disjoint) of lengths $T_1 = 2^n - 1$ and $T_2 = (2^n - 1)n/g$, is exactly $(2^n - 1)/g$ or $\lfloor (2^n - 1)/g \rfloor - 1$.*

Proof. (a) Let the chain begin at (m, X) . Consider a companion pair $(m + i, \alpha^i X)$ and $(m + j, \alpha^j X)$, $i < j$. By distance between the nodes of a companion pair we will mean the quantity $j - i$. Because these nodes are companions,

$$(3.19) \quad i \equiv j \pmod{n}$$

and

$$(3.20) \quad \alpha^i X + \alpha^j X = \beta_0.$$

From (3.20) one gets

$$(3.21) \quad \alpha^{j-i} X + X = \alpha^{-i} \beta_0.$$

Consider now another distinct companion node pair $(m + i', \alpha^{i'} X)$ and $(m + j', \alpha^{j'} X)$, $i' < j'$. For this pair, one could similarly show that

$$(3.22) \quad \alpha^{j'-i'} X + X = \alpha^{-i'} \beta_0.$$

If $i, i' < 2^n - 1$, then (3.21) and (3.22) imply that the distance $j' - i'$ must be different from $j - i$; or else α^i would equal $\alpha^{i'}$, which is impossible for $i, i' < 2^n - 1$. Further, from (3.19), the distances must always take values that are multiples of n . Therefore there are at most $\lfloor (T - 1)/n \rfloor$ companion node pairs when $T \leq 2^n - 1$.

For larger values of T , either the first node of the pair is within the first $2^n - 1$ nodes of the chain, or the pair is entirely confined to the last $T - (2^n - 1)$ elements of

the chain. Since the number of companion node pairs of these two kinds are at most $\lfloor (T - 1)/n \rfloor$ and $\Gamma(T - (2^n - 1))$, respectively, we get the stated bound.

(b) Let the two chains begin at (m, X) and (m', X') . Consider a companion pair $(m + i, \alpha^i X)$ and $(m' + j, \alpha^j X')$ across the chains. By arguments similar to part (a), $j - i$ must be a multiple of n . Since $-T_1 \leq j - i \leq T_2$, we get the specified bound on the number of pairs.

(c) Let $T_1 = 2^n - 1$ and $T_2 = (2^n - 1)n/g$. Denote by g_1 and g_2 , the partition indices of the nodes in the two chains. Let (m, α^i) be a typical node of the first chain, $0 \leq i \leq 2^n - 2$. We want to determine if its companion (m, α^j) is in the second chain. Since the second chain occupies a whole partition, the companion will belong to it if its partition index is g_2 . When $\alpha^i = \beta_0$, the companion, $(m, 0)$, is clearly not a member of the second chain. When $\alpha^i \neq \beta_0$, from the companion relationship between nodes (m, α^i) and (m, α^j) , one gets $\alpha^{j-i} = 1 + \alpha^{-i}\beta_0$. Thus for each of the $2^n - 2$ values of i (excluding the one corresponding to $\alpha^i = \beta_0$), $j - i \pmod{(2^n - 1)}$ takes a different value. Consequently it assumes all the values from 0 to $2^n - 2$ except one. Now, one has

$$\begin{aligned} \text{partition index of } (m, \alpha^j) &= j - m \pmod{g} \\ &= ((j - i) \pmod{(2^n - 1)} + g_1) \pmod{g}. \end{aligned}$$

Since $(j - i) \pmod{(2^n - 1)}$ takes all values from 0 to $2^n - 2$ except one, the partition index of (m, α^j) will assume value g_2 exactly $(2^n - 1)/g$ or $\lfloor (2^n - 1)/g \rfloor - 1$ times, showing that there exist exactly these many companion node pairs between the two chains. \square

Note that the two chains of part (c) of Lemma 3.5 may overlap, unlike those of part (b).

We now state Theorem 3.6 relating to mapping cycles of all allowed lengths larger than $2n$ to B_n . (Actually, odd lengths between n and $2n$ are also covered by this result.) Because of Lemma 3.5, this theorem only needs to prove the existence of sufficient number of external node pairs to attach to the cycle constructed as per Theorem 3.3.

THEOREM 3.6 (arbitrary length cycles of lengths $\geq n$).

(a) For odd n , a cycle of any length L , $n \leq L \leq n2^n$, excluding even values of L less than $2n$, can be mapped to B_n with dilation 1.

(b) For even n , a cycle of any even length L , $n \leq L \leq n2^n$, can be mapped to B_n with dilation 1.

Proof. If $n|L$, the desired cycle is already addressed in Theorem 3.3. For other L values, first form a cycle of length Kn using Theorem 3.3, where K is the largest possible number such that $Kn < L$ and $L - Kn$ is even. We will call this cycle the primary cycle. By adding up to $n - 1$ pairs of outside nodes to the primary cycle we get the required cycle of length L . (If $K = 2^n - 1$, one needs to add only up to $\lfloor n/2 \rfloor$ pairs to get the largest required length.)

We will show that there are sufficient number of nodes in the primary cycle without their companion nodes. Following the method of Figure 3.2, one can attach a pair of external nodes at each of these nodes. We first prove the theorem when $g = 1$ (Case 1). Cases 2, 3, and 4 deal with $g > 1$, and assume $n \geq 6$ since it is the smallest n for which $g \neq 1$. The parameter that distinguishes these cases is t , the number of smaller cycles C_1, C_2, \dots, C_t that are merged as in Theorem 3.3 to obtain the primary cycle.

Case 1. $g = 1$. If $K = 1$, then each node of the primary cycle is without its companion node, and each new added node pair is distinct. The first part of this is true because companion node pairs must have a distance of at least n . To see the second part, compare the pairs added at two consecutive cycle nodes, say, (m, X) and $(m + 1, \alpha X)$. The outside pair added at the first node is $(m, X + \beta_0) \rightarrow (m + 1, \alpha X + \beta_{n-1})$, and the one added at the second node is $(m + 1, \alpha X + \beta_0) \rightarrow (m + 2, \alpha^2 X + \beta_{n-1})$. Clearly all these four new nodes are distinct. In a similar fashion, one can show that up to $n - 1$ new distinct pairs of nodes may be added to the cycle.

When $K = 2^n - 1$, the primary cycle consists of all the nodes (m, X) , $0 \leq m < n$, $X \in GF(2^n)$, $X \neq 0$. In this case, at each node (m, β_0) in the cycle, one can attach an outside pair $(m, 0) \rightarrow (m + 1, 0)$. Distinctness of the new pairs can be ensured by using only even values of m . In this manner, up to $\lfloor n/2 \rfloor$ pairs of outside nodes may be attached to the cycle to achieve the required length.

Unfortunately, when $2 \leq K \leq 2^n - 2$, the second node of a pair may, at times, turn out to be the same as the first node of another pair. We therefore would not be able to add both these pairs to the cycle at the same time. However, if we have $2(n - 1)$ nodes without their companion nodes, we can guarantee adding at least half of the outside node pairs, i.e., $(n - 1)$ pairs. We will now show that the primary cycle, indeed, contains $2(n - 1)$ nodes without their companion nodes. We will prove this for $Kn \leq n(2^n - 1)/2$ only. For larger Kn values, one may consider the complementary cycle of length $n(2^n - 1) - Kn$ and prove similarly the existence of at least $2(n - 1)$ nodes therein, which have their companion nodes outside, i.e., in the original cycle of length Kn .

Using Lemma 3.5(a) we can find the maximum number of companion node pairs within the cycle of length Kn . Subtracting these nodes from the total number of nodes in the cycle, we find that at least

$$(3.23) \quad Kn - 2\Gamma(Kn)$$

cycle nodes have companion nodes outside the cycle. For $Kn \leq 2^n - 1$, use of Lemma 3.5(a) in (3.23) gives

$$\begin{aligned} \text{number of nodes with external companions} &\geq Kn - 2(K - 1) \\ &= (2n - 2) + (n - 2)(K - 2) \\ &> 2n - 2. \end{aligned}$$

This proves that there are sufficient number of companion node pairs in the primary cycle where new node pairs may be attached.

To prove the result for $Kn > 2^n - 1$ by mathematical induction, assume its truth for length $Kn - (2^n - 1)$; i.e., assume that

$$[Kn - (2^n - 1)] - 2\Gamma(Kn - (2^n - 1)) \geq 2n - 2.$$

Recall that we need only to prove the result for $Kn \leq n(2^n - 1)/2$. One now gets for the primary cycle of length Kn ,

$$\begin{aligned} \text{no. of nodes with external companions} &\geq Kn - 2[\lfloor (Kn - 1)/n \rfloor + \Gamma(Kn - (2^n - 1))] \\ &\geq (2n - 2) + (2^n - 1) - 2(K - 1) \\ &\geq (2n - 2) + 2 \\ &> 2n - 2. \end{aligned}$$

Case 2. $g > 1$ and $t < g$. If the primary cycle length $Kn \leq 2^n - 1$, the situation is similar to that of Case 1. For $Kn > 2^n - 1$, we proceed as follows. Since $t < g$, at least one partition representing a chain of length $n(2^n - 1)/g$ is not used in the primary cycle. Consider this chain and a chain of length $2^n - 1$ in C_1 of nodes used in the primary cycle. From Lemma 3.5(c), there are at least $\lfloor (2^n - 1)/g \rfloor - 1$ companion node pairs across them. But note that for $n \geq 6$, $2^n - 1 > n^2$. Further, $g \leq n/2$. Therefore $\lfloor (2^n - 1)/g \rfloor - 1 > 2n - 2$ showing that at least $2n - 2$ companion node pairs exist between the primary cycle and the remaining nodes.

Case 3. $g > 1$, $t = g$ and cycles C_1, C_2, \dots, C_{t-1} have lengths $n(2^n - 1)/g$. If the number of nodes left out of the primary cycle is less than or equal to $2^n - 1$, then one can prove this case in a manner similar to Case 1 except that the focus will now be on the nodes that are *not* in the cycle rather than those that are part of the cycle. But the final consequence is the same: there are enough companion pairs between the nodes in the cycle and those outside. If the number of nodes outside the primary cycle is more than $2^n - 1$, then a chain of length $2^n - 1$ of these outside elements will have at least $\lfloor (2^n - 1)/g \rfloor - 1$ companion nodes within C_1 . (Lemma 3.5(c)). This number is greater than $2n - 2$ (for $n \geq 6$) showing that there are enough companion node pairs between the primary cycle and the outside nodes.

Case 4. $g > 1$, $t = g$ and cycles C_1, C_2, \dots, C_{t-2} have lengths $n(2^n - 1)/g$. First note that because g is odd, when $g \neq 1$, $g \geq 3$. Thus, in this case, cycle C_1 of $n(2^n - 1)/g$ nodes is part of the primary cycle. Further, from the construction in Theorem 3.3, for this case to exist, the number of unused nodes in partitions of C_{t-1} and C_t —call them R_1 and R_2 , respectively—must satisfy

$$(3.24) \quad R_1 + R_2 > (2^n - 1)(n/g - 1).$$

Without loss of generality, let $R_1 \leq R_2$. Clearly, both these sets of unused elements form chains. If either of these chains has at least $2^n - 1$ elements, then at least $(2^n - 1)/g$ of them will have companion nodes in C_1 . Thus as in Case 3, there are enough companion node pairs between the nodes of the companion cycle and those outside. On the other hand, if both $R_1, R_2 < 2^n - 1$, then from Lemma 3.5(a) and (b), one can see that there are at most $\lfloor (R_1 - 1)/n \rfloor$ and $\lfloor (R_2 - 1)/n \rfloor$ companion node pairs within these chains and $\lfloor (R_1 + R_2)/n \rfloor + 1$ across the two chains. Since there are a total of $R_1 + R_2$ nodes in these two chains, the rest of their nodes must have companions in the primary cycle. Since R_1 and R_2 are multiples of n , $\lfloor (R_1 - 1)/n \rfloor = (R_1/n) - 1$ and $\lfloor (R_2 - 1)/n \rfloor = (R_2/n) - 1$. Using (3.24) one thus gets that the number of companion node pairs between the primary cycle and those outside to be at least

$$(3.25) \quad \begin{aligned} (R_1 + R_2) - 2[R_1/n - 1 + R_2/n - 1 + (R_1 + R_2)/n + 1] &= (R_1 + R_2)(1 - 4/n) + 2 \\ &> (2^n - 1)(n/g - 1)(1 - 4/n) + 2 \\ &\geq (2^n - 1)/3 + 2 \\ &> (2n - 2). \end{aligned}$$

The simplification in the third line of (3.25) is based on the fact that $(n/g) \geq 2$ and $(1 - 4/n) \geq 1/3$ for $n \geq 6$, while last line of (3.25) is true for any $n \geq 6$. \square

Theorem 3.6 proves that there are sufficient number of nodes in the primary cycle where one can attach outside node pairs to obtain any desired length cycle as long as this length can be expressed as $Kn + 2t$. Construction of such a cycle is rather straightforward; once the primary cycle is obtained as per Theorem 3.3, one only needs to identify the required number of cycle nodes whose companions are outside

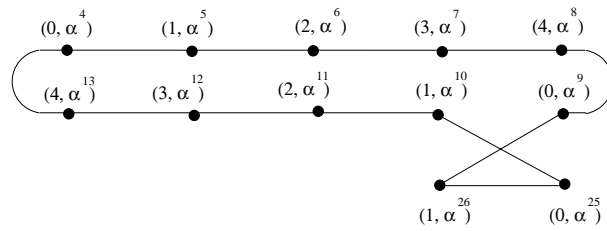


FIG. 3.3. A length 12 cycle mapped to $Z_5 \times GF(2^5)$ with dilation 1.

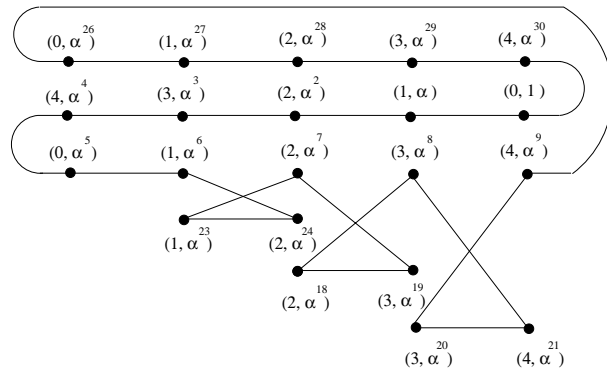


FIG. 3.4. A length 21 cycle mapped to $Z_5 \times GF(2^5)$ with dilation 1.

$$\begin{aligned}
 &(0, 11101) \rightarrow (1, 11100) \rightarrow (2, 11100) \rightarrow (3, 00011) \rightarrow \\
 &(4, 11000) \rightarrow (0, 01000) \rightarrow (1, 01000) \rightarrow (0, 01001) \rightarrow \\
 &(1, 01001) \rightarrow (2, 01001) \rightarrow (3, 01101) \rightarrow (4, 01101) \rightarrow (0, 11101)
 \end{aligned}$$

$$\begin{aligned}
 &(0, 00100) \rightarrow (1, 00101) \rightarrow (2, 00111) \rightarrow (3, 00111) \rightarrow (4, 01111) \rightarrow \\
 &(0, 11111) \rightarrow (1, 11110) \rightarrow (2, 11110) \rightarrow (3, 11110) \rightarrow (4, 11110) \rightarrow \\
 &(0, 01110) \rightarrow (1, 01110) \rightarrow (2, 01110) \rightarrow (1, 01100) \rightarrow (2, 01100) \rightarrow \\
 &(3, 01000) \rightarrow (2, 01000) \rightarrow (3, 01100) \rightarrow (4, 01100) \rightarrow (3, 00100) \rightarrow \\
 &(4, 00100) \rightarrow (0, 00100)
 \end{aligned}$$

FIG. 3.5. Cycles of length 12 and 21 in B_5 .

the cycle. Finding a small number of these nodes is a relatively simple process. New node pairs may be attached at these nodes as in Figure 3.2 to get the desired cycle.

We illustrate this process by constructing length 12 and 21 cycles in B_5 . The final construction is shown in Figures 3.3 and 3.4. To construct the cycle of length 12, one first uses Theorem 3.3 to create a primary cycle of length 10 (beginning at $(0, \alpha^4)$). Since $(0, \alpha^9)$ belongs to the cycle, but not its companion $(0, \alpha^9 + \beta_0) = (0, \alpha^{25})$, we add the indicated pair to get length 12 cycle. (Note that companions of none of the nodes in this length 10 cycle are present in it. Thus one could have added a new node pair at any of these nodes.) Similarly, a length 21 cycle is obtained by first creating a primary cycle of length 15 (beginning at $(0, \alpha^{26})$) and then adding pairs of new nodes at $(1, \alpha^6)$, $(2, \alpha^7)$, and $(3, \alpha^8)$. These cycles translated to binary notation using Table 2.3 are shown in Figure 3.5.

$$\begin{aligned}
 (0, 0) & \qquad \qquad \qquad \rightarrow (1, 0) & \qquad \qquad \qquad \cdots \rightarrow (K, 0) & \qquad \qquad \rightarrow \\
 (K - 1, \beta_0) & \qquad \qquad \rightarrow (K - 2, \beta_0(\alpha^{-1} + 1)) & \qquad \cdots \rightarrow (0, \beta_0 \sum_{i=0}^{K-1} \alpha^{-i}) & \rightarrow \\
 (1, \beta_0 \sum_{i=-1}^{K-2} \alpha^{-i}) & \qquad \rightarrow (2, \beta_0 \sum_{i=-2}^{K-3} \alpha^{-i}) & \qquad \cdots \rightarrow (K, \beta_0 \sum_{i=-K}^{-1} \alpha^{-i}) & \rightarrow \\
 (K - 1, \beta_0 \sum_{i=-(K-1)}^{-1} \alpha^{-i}) & \rightarrow (K - 2, \beta_0 \sum_{i=-(K-2)}^{-1} \alpha^{-i}) & \cdots \rightarrow (1, \beta_0 \sum_{i=-1}^{-1} \alpha^{-i}) & \rightarrow (0, 0)
 \end{aligned}$$

FIG. 3.6. Length $4K$ cycle mapping to B_n ($K < n$).

Unfortunately, Theorem 3.6 does not cover all the cycles that can possibly be mapped to B_n . Particularly, when n is odd, Theorem 3.6 does not provide constructions of cycles of even lengths less than $2n$ and when n is even, it excludes even lengths less than n . We now illustrate a technique to cover these cases. Using this technique, one can easily map cycles of even lengths less than $4n$ (except possibly lengths 6 and 10) to B_n with dilation 1. This implies that all the cycles that are not proved to be impossible in Theorem 3.1 can indeed be mapped to B_n . This final result about cycles is stated in Theorem 3.7.

THEOREM 3.7 (comprehensive cycle mapping). *One can map to B_n all cycles, except those identified in Theorem 3.1, with dilation 1.*

Proof. Because of Theorem 3.6, the only cycles we need to map to B_n are those with even lengths less than $2n$. We can use the following construction for even lengths less than $4n$.

If $L = 4K$, $K < n$, construct the cycle as follows.

Start from node $(0, 0)$. Let (m, X) denote the current node on the cycle. Use K times link $(m, X) \rightarrow (m+1, \alpha X)$. Then use K times link $(m, X) \rightarrow (m-1, \alpha^{-1}X + \beta_0)$. Follow it K times with link $(m, X) \rightarrow (m + 1, \alpha X)$. Finally, K times, travel along $(m, X) \rightarrow (m - 1, \alpha^{-1}X + \beta_0)$. This will bring you back to the starting node $(0, 0)$. Figure 3.6 shows this length $4K$ cycle.

One can see from the connectivity of B_n , shown in Figure 2.1, that the consecutive nodes in the cycle above are indeed connected. We need only to show that they are distinct. The only two nodes in the cycle with the first index of the label 0 are $(0, 0)$ and $(0, \beta_0 \sum_{i=0}^{K-1} \alpha^{-i})$. Clearly these are not the same since $K < n$. For the same reason, the two nodes with the first index of label being K , $(K, 0)$ and $(K, \beta_0 \sum_{i=-K}^{-1} \alpha^{-i})$ are distinct. The four cycle nodes with the same first index m , $0 < m < K$, are $(m, 0)$, $(m, \beta_0 \sum_{i=0}^{K-1-m} \alpha^{-i})$, $(m, \beta_0 \sum_{i=-m}^{K-1-m} \alpha^{-i})$, and $(m, \beta_0 \sum_{i=-m}^{-1} \alpha^{-i})$. One can see that the second indices of these four nodes are distinct because $K < n$ and α , being a primitive element of $GF(2^n)$, cannot satisfy any equation of degree less than n . Thus one can map all cycles of length $4K$, $K < n$, to B_n with dilation 1.

If $L = 4K + 2$ and $K > 2$, one can add a pair $(2, \beta_{n-1}) \rightarrow (1, \beta_0)$ of new nodes between the cycle nodes $(1, 0) \rightarrow (2, 0)$ as in Figure 3.2. Thus, replacing the first three elements in the cycle of Figure 3.6 by the five elements: $(0, 0) \rightarrow (1, 0) \rightarrow (2, \beta_{n-1}) \rightarrow (1, \beta_0) \rightarrow (2, 0)$, one gets a new cycle of length $4K + 2$. It is easy to verify that the new elements were indeed absent from the original cycle when $K > 2$. Thus one can map all cycles of length $4K + 2$, $2 < K < n$, to B_n with dilation 1. The only even-length cycles less than $4n$ excluded by this procedure have lengths 6 and 10. \square

The construction of Theorem 3.7 may be illustrated by mapping a cycle of length 14 to B_5 . Since $14 = 12 + 2$, we first construct a cycle of length 12 and then add a new pair to it. The final cycle and its binary translation are shown in Figures 3.7 and 3.8.

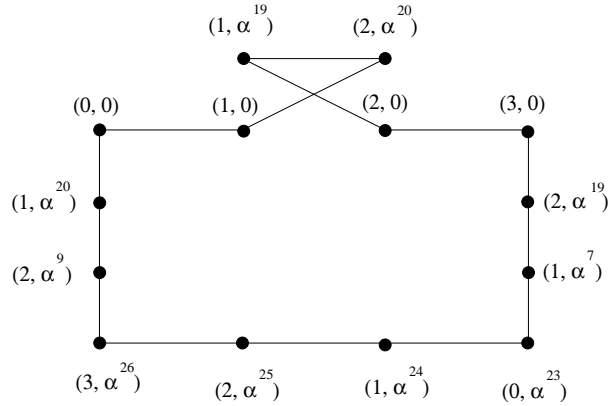


FIG. 3.7. A length 14 cycle mapped to $Z_5 \times GF(2^5)$.

$$\begin{aligned}
 (0, 00000) &\rightarrow (1, 00000) \rightarrow (2, 00010) \rightarrow (1, 00010) \rightarrow (2, 00000) \rightarrow \\
 (3, 00000) &\rightarrow (2, 00100) \rightarrow (1, 00110) \rightarrow (0, 00110) \rightarrow (1, 00111) \rightarrow \\
 (2, 00101) &\rightarrow (3, 00001) \rightarrow (2, 00001) \rightarrow (1, 00001) \rightarrow (0, 00000)
 \end{aligned}$$

FIG. 3.8. A length 14 cycle in B_5 .

4. Mapping binary trees on the butterfly. This section presents two results about mapping trees to B_n . We first show that it is possible to map multiple nonoverlapping balanced binary trees with dilation 1 to this network (Theorem 4.1). Next, by combining these trees, we show that one can map the largest possible binary tree to B_n with a slightly higher dilation (Theorem 4.3). With the new model for B_n , both these tasks are relatively simple.

In the discussion that follows, we restrict ourselves to balanced binary trees. By level of a node in the tree we will mean its distance from the root. An m -level tree has nodes in levels $0, 1, \dots, m - 1$. Each parent has exactly two children. Thus the m -level tree has 2^{m-1} leaves and $2^m - 1$ total nodes. All the logarithms are assumed to be base 2.

THEOREM 4.1 (multiple trees). *One can map n nonoverlapping n -level binary trees to B_n with dilation 1.*

Proof. Choose node (i, β_{n-1}) to be the root of the i th tree, $0 \leq i < n$. Construct each tree by the simple rule that the children of any node (j, x) are nodes $(j + 1, \alpha x)$ and $(j + 1, \alpha x + \beta_{n-1})$.

Clearly, both the children in each tree node are connected to the parent by a direct edge (see Figure 2.1). We need only to prove that all the nodes in these trees are distinct. Note that, because of the way the trees are constructed, the nodes in level j of i th tree have labels $((i + j) \bmod n, \beta_{n-1} f(\alpha))$ where $f(\alpha)$ are distinct binary polynomials of α of degree j . Clearly, nodes on different levels of a tree are distinct since their first indices are unequal. Similarly, all nodes on the same level of a tree are also distinct; otherwise, their second indices, $\beta_{n-1} f_1(\alpha)$ and $\beta_{n-1} f_2(\alpha)$ will be equal, implying that $f_1(\alpha) - f_2(\alpha)$, a polynomial in α of degree less than n , equals zero. This is impossible since α is a primitive element of $GF(2^n)$. Finally, suppose a node $((i_1 + j_1) \bmod n, \beta_{n-1} f_1(\alpha))$ of tree i_1 is identical to the node $((i_2 + j_2) \bmod n, \beta_{n-1} f_2(\alpha))$ of a different tree i_2 . Since the first indices of the two nodes are the same, $j_1 \neq j_2$. Thus, equality of the second index implies two polynomials

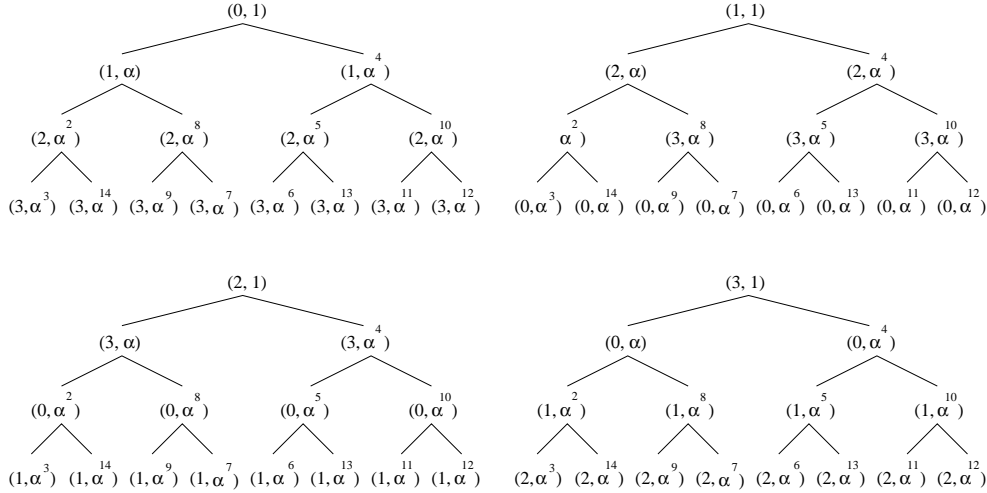


FIG. 4.1. Four nonoverlapping 4-level trees mapped to B_4 with dilation 1.

$f_1(\alpha)$ and $f_2(\alpha)$ of unequal degrees (and each less than n) are equal. This is again impossible in $GF(2^n)$. Therefore all nodes in all trees are distinct. \square

Bhatt et al. [2] have previously proved the same result stated in Theorem 4.1. However, our construction is based on the butterfly modeled as $Z_n \times GF(2^n)$ and is necessary to obtain the mapping of the largest tree as given in Theorem 4.3.

Note that the mapping given in Theorem 4.1 is optimum in the sense that it describes the maximum number of nonoverlapping n -level binary trees that may be mapped to B_n . Clearly the unused n nodes, $\{(i, 0) \mid 0 \leq i < n\}$, do not support another n -level tree. Figure 4.1 shows mapping of four disjoint trees to B_4 .

We now focus on mapping the single largest binary tree to B_n . Such a tree would have $n + \lceil \log n \rceil$ levels and would use $2^{\lceil \log n \rceil} 2^n - 1$ nodes of B_n . For n values which are powers of 2, such a tree would span all but one node of B_n . For other values of n , this is the largest tree that may be mapped to B_n , because increasing the number of tree levels even by 1 will imply more nodes than the number of nodes of B_n .

We create this tree by first designing the top $\lceil \log n \rceil$ levels using nodes unused in Theorem 4.1. Then two n -level trees generated as in Theorem 4.1 are attached at each leaf of this tree. Since a $\lceil \log n \rceil$ -level tree has at most $n/2$ leaves, the n nonoverlapping trees obtained in Theorem 4.1 suffice. The top tree, however, needs to be designed carefully since its leaves should be able to connect (with low dilation) to the roots of the lower n -level trees which are very specific. Recall that the only nodes unused in Theorem 4.1 are $(i, 0)$, $0 \leq i < n$. Lemma 4.2 provides the required mapping of the $\lceil \log n \rceil$ -level tree to these nodes.

LEMMA 4.2. *One can map a $\lceil \log n \rceil$ -level binary tree to nodes $(i, 0)$, $0 \leq i < n$, of B_n such that all its leaves are mapped to $(i, 0)$ with odd i . Further, the dilation of this mapping is $2^{\lceil \log n \rceil} / 4$.*

Proof. Let n' denote $\lceil \log n \rceil$. Number the tree levels 0 through $\lceil \log n \rceil - 1$ with the root at level 0. Map the tree root to node $(2^{n'-1}, 0)$. Map the children of a parent $(i, 0)$ at level l of the tree to nodes $(i - 2^{n'-2-l}, 0)$ and $(i + 2^{n'-2-l}, 0)$.

One can verify that this procedure maps 2^l tree vertices at level l to nodes $(2^{n'-1-l}p, 0)$, with odd p , $1 \leq p \leq 2^{l+1} - 1$. Thus all tree nodes are mapped to distinct nodes of B_n . The leaves of this tree are mapped to $(p, 0)$ as specified.

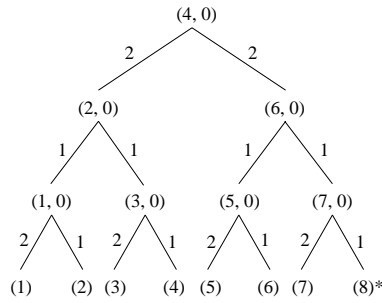


FIG. 4.2. The top of the tree in B_n , $8 \leq n < 16$ (with dilation marked on each edge). The last row contains the roots of n -level trees. (For brevity, a root (i, β_{n-1}) is shown only as (i) . Further, $(8)^*$ denotes root $(0, \beta_{n-1})$ when $n = 8$.)

Further, the paths from the image of a parent at level l to images of its children are of length $2^{n'-2-l}$. The longest of these paths gives the specified dilation of the mapping. \square

The mapping of Lemma 4.2 exhibits a congestion of $(n' - 1)$; i.e., $(n' - 1)$ paths corresponding to tree edges pass through a single edge B_n . To see this, first note that all the paths in B_n corresponding to edges of any single tree level are disjoint. Thus the congestion cannot exceed $(n' - 1)$. One can also show that the B_n edge between nodes $(\lfloor 2^{n'}/3 \rfloor, 0)$ and $(\lfloor 2^{n'}/3 \rfloor + 1, 0)$ carries paths corresponding to a tree edge at every level. Thus this B_n edge has a congestion of $(n' - 1)$.

We now combine Theorem 4.1 and Lemma 4.2 to derive the central result of this section.

THEOREM 4.3 (tree mapping). *One can map a binary tree of $n + \lfloor \log n \rfloor$ levels to B_n with dilation 2 if $n < 16$, 3 if $16 \leq n < 32$, 4 if $32 \leq n < 64$, and $2^{\lfloor \log n \rfloor} / 4$ if $n \geq 64$.*

Proof. We map the top $\lfloor \log n \rfloor$ levels of the tree as in Lemma 4.2. Note that the leaves of this $\lfloor \log n \rfloor$ -level tree are mapped to $(i, 0)$ for odd i . At each leaf $(i, 0)$ we attach two n -level trees generated according to Theorem 4.1 with roots at (i, β_{n-1}) and $(i + 1, \beta_{n-1})$. Obviously all the nodes in the resultant $(n + \lfloor \log n \rfloor)$ -level tree are distinct.

Further, the roots of the n -level tree are at distances 2 and 1 from the corresponding leaves of the $\lfloor \log n \rfloor$ -level tree. The dilation within the lower n -level trees is 1. Therefore the overall dilation of the tree is dictated by the dilation within the top $\lfloor \log n \rfloor$ levels which, according to Lemma 4.2, is $2^{\lfloor \log n \rfloor} / 4$.

As an example, the $\lfloor \log n \rfloor$ -level tree for $8 \leq n < 16$ is shown in Figure 4.2. The top three levels of this tree are generated from Lemma 4.2, and the fourth level shows the roots of n -level trees. When $16 \leq n < 64$, one can use the top trees shown in Figures 4.3 and 4.4 rather than the ones obtained from Lemma 4.2. These trees are obtained by shuffling certain nodes of the tree obtained from Lemma 4.2 to reduce the dilation. Since in the new $\lfloor \log n \rfloor$ -level trees the leaves are not necessarily at $(i, 0)$ with odd i 's, the choice of the n -level trees attached to each leaf is also different. The last rows of these figures specify the roots of the n -level trees to be attached to each leaf. \square

The tree mapping strategy presented here is interesting because of its extreme simplicity. The bottom n levels of the tree using $2^{\lfloor \log n \rfloor} (2^n - 1)$ nodes are mapped in a very systematic manner. The top $\lfloor \log n \rfloor$ levels of the tree using fewer than n nodes may also be algorithmically created using Lemma 4.2. The overall dilation is decided

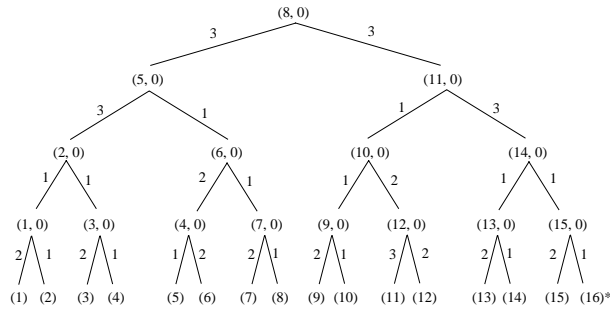


FIG. 4.3. A better top for a tree in B_n , $16 \leq n < 32$ (with dilation marked on each edge). The last row contains the roots of the n -level trees. (For brevity, a root (i, β_{n-1}) is shown only as (i) . Further, $(16)^*$ denotes root $(0, \beta_{n-1})$ when $n = 16$.)

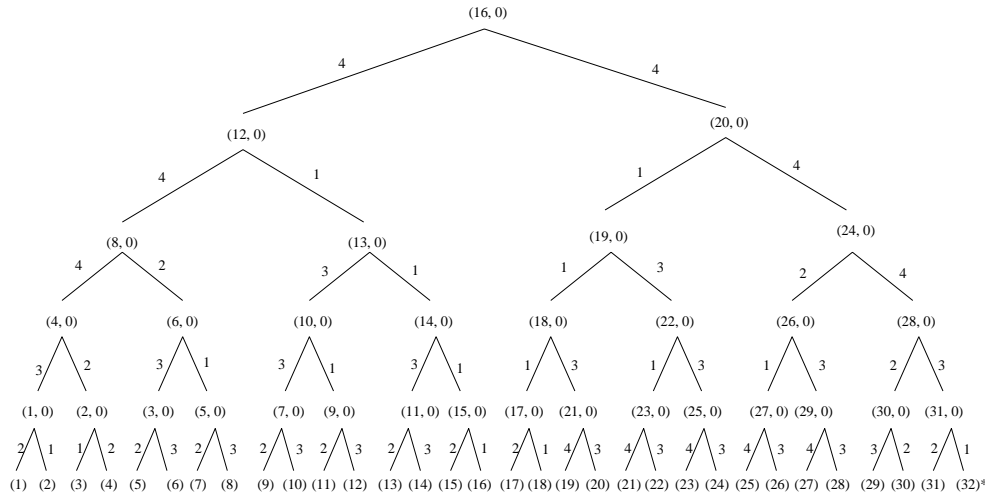


FIG. 4.4. A better top for a tree in B_n , $32 \leq n < 64$ (with dilation marked on each edge). The last row contains the roots of the n -level trees. (For brevity, a root (i, β_{n-1}) is shown only as (i) . Further, $(32)^*$ denotes root $(0, \beta_{n-1})$ when $n = 32$.)

by the dilation within these top levels. The dilation may be reduced by modifying the top tree. Fortunately, for every set of n values between two consecutive powers of 2, one needs to design a single top tree. We have altered the top tree to limit dilation to 3 when $16 \leq n < 32$ and to 4 when $32 \leq n < 64$. Such alterations may sometimes increase the congestion. The congestions of the top trees of Figures 4.2 and 4.3 are 2 and 3, respectively. These values are identical to the congestions of the same size top trees built using Lemma 4.2. But for the range $16 \leq n < 32$, the congestion of the top tree in Figure 4.4 is 6 as against the congestion of 4 for the top tree built using Lemma 4.2. Note that this increase in congestion yields a decrease of dilation from 8 to 4.

5. Conclusion. This paper has given a new model for the wrap-around butterfly networks and has demonstrated its utility to obtain mappings of cycles and trees.

Our results about cycle mappings are presented in Table 5.1. We have identified all cycles that could be subgraphs of butterfly networks and have provided their

TABLE 5.1
The existence of cycles as subgraphs of B_n .

Cycle length L	$3 \leq L < n$	$n \leq L < 2n$	$2n \leq L \leq n2^n$
n odd L even	Theorem 3.7*	Theorem 3.7 *	Theorem 3.6
n odd L odd	Impossible	Theorem 3.6	Theorem 3.6
n even L even	Theorem 3.7*	Theorem 3.6	Theorem 3.6
n even L odd	Impossible	Impossible	Impossible

* When $n = 5$ or $n \geq 7$, cycle of length 6 is not possible.
 When $n = 7, 9$ or $n \geq 11$, cycle of length 10 is not possible.

TABLE 5.2
Mapping a tree of $2^{n+\lfloor \log n \rfloor} - 1$ nodes to a butterfly architecture.

Reference	Architecture size	Dilation	Conditions
Bhatt et al. [2]	$(n+3)2^{n+3}$	4	
Gupta et al. [6]	$(n+1)2^{n+1}$	4	even n
Gupta et al. [6]	$(n+2)2^{n+2}$	2	even n
This paper	$n2^n$	2	$n < 16$
This paper	$n2^n$	3	$16 \leq n < 32$
This paper	$n2^n$	4	$32 \leq n < 64$
This paper	$n2^n$	$2^{\lfloor \log n \rfloor} / 4$	

mappings. We give two procedures for mapping cycles to B_n : one that is applicable to cycles of even lengths less than $4n$ and the other for larger lengths. In the first case, the cycle is established directly. In the second, one sets up a cycle of length divisible by n and then augments it with a small number of node pairs to make up the required length. Earlier results [11] about cycle mappings on B_n had identified $O(2^n)$ cycle subgraphs each with a different length, whereas we provide $O(n2^n)$ cycle subgraphs of different lengths.

Our results about tree mappings are listed in Table 5.2. Using our methods one can map the largest possible balanced binary tree to B_n , $n < 16$, with a small dilation of 2. One may note that B_{15} , the biggest butterfly network to which this result applies, has almost half a million nodes. We also give maximal tree mappings in larger butterflies with bounded dilation. For $16 \leq n < 32$, the dilation is 3, and for $32 \leq n < 64$, it is 4. Thus even though the tree mapping results presented here are asymptotically poor as compared to the earlier work [2, 6] (dilation and congestion $O(n)$ as against $O(1)$), for practical network sizes of up to $n = 64$, they have low dilation and congestion. Further, unlike the earlier mappings, ours does not require a larger size butterfly to ensure the one-to-one (load = 1) mapping. Thus to map the same tree, we use a butterfly with at most half the nodes as before. Our trees have unit dilation in their lower n levels. A larger dilation may be present only within the top $\lfloor \log n \rfloor + 1$ levels that employ fewer than n nodes.

The simplicity of the mappings obtained here is essentially due to our identification of the butterfly network with the direct product of a group and a finite field. We have shown that in the context of this model, the network connectivity may be expressed as an algebraic relationship between the node labels. One may then employ the powerful tools of abstract algebra to explore the structural properties of the network. Even though we have limited our investigation here to certain mappings, we

believe that these methods hold a lot of promise for other aspects of these networks as well.

Acknowledgment. The authors wish to thank anonymous referees for comments and suggestions that improved the paper tremendously.

REFERENCES

- [1] E. R. BERLEKAMP, *Algebraic Coding Theory*, McGraw-Hill, New York, 1968.
- [2] S. N. BHATT, F. R. K. CHUNG, J. HONG, F. T. LEIGHTON, B. OBRENIC, A. L. ROSENBERG, AND E. J. SCHWABE, *Optimal emulation by butterfly-like networks*, J. ACM, 43 (1996), pp. 293–330.
- [3] G. CHEN AND C. M. LAU, *Comments on “A new family of Cayley graph interconnection networks of constant degree four”*, IEEE Trans. Parallel Distrib. Syst., 8 (1997), pp. 1299–1300.
- [4] R. FELDMANN AND W. UNGER, *The cube-connected cycle is a subgraph of the butterfly network*, Parallel Process. Lett., 2 (1992), pp. 13–19.
- [5] C. T. GRAY, W. LIU, T. HUGHES, AND R. CAVIN, *The design of a high-performance scalable architecture for image processing applications*, in Proceedings of the 1990 International Conference on Application Specific Array Processors, IEEE, Piscataway, NJ, 1991, pp. 722–733.
- [6] A. GUPTA AND S. E. HAMBRUSCH, *Embedding complete binary trees into butterfly networks*, IEEE Trans. Comput., 40 (1991), pp. 853–863.
- [7] F. T. LEIGHTON, *Introduction to Parallel Algorithms and Architectures: Arrays, Trees, Hypercubes*, Morgan Kaufman, San Mateo, CA, 1992.
- [8] W. LIN, T. SHEU, C. R. DAS, T. FENG, AND C. WU, *Fast data selection and broadcast on the butterfly network*, in Proceedings of the International Workshop on Future Trends of Distributed Computing Systems in the 1990s, 1998, pp. 65–72.
- [9] A. RANADE, *Optimal speedup for backtrack search on a butterfly network*, Math. Systems Theory, 27 (1994), pp. 85–102.
- [10] J. H. REIF AND S. SEN, *Randomized algorithms for binary search and load balancing on fixed connection networks with geometric applications*, SIAM J. Comput., 23 (1994), pp. 633–651.
- [11] A. L. ROSENBERG, *Cycles in Networks*, Technical report 91–20, University of Massachusetts, Amherst, 1993.
- [12] E. J. SCHWABE, *Constant-slowdown simulations of normal hypercube algorithms on the butterfly network*, Inform. Process. Lett., 45 (1993), pp. 295–301.
- [13] P. VADAPALLI AND K. SRIMANI, *A new family of Cayley graph interconnection networks of constant degree four*, IEEE Trans. Parallel Distrib. Syst., 17 (1996), pp. 26–32.

WDM SWITCHING NETWORKS, REARRANGEABLE AND NONBLOCKING $[w, f]$ -CONNECTORS*

HUNG Q. NGO[†]

Abstract. We propose a framework to analyze and compare wavelength division multiplexed (WDM) switching networks qualitatively and quantitatively. The framework not only helps analyze and compare the complexity of WDM switching networks but also explains interesting properties of different designs. Then several important problems arising from this idea are addressed, and complexity bounds are derived. We also give several applications of the proposed model, including explicit constructions of nonblocking WDM switching fabrics.

Key words. wavelength division multiplexing, switching networks, $[w, f]$ -connectors

AMS subject classifications. 05C75, 05C90, 05C35

DOI. 10.1137/S0097539704442714

1. Introduction. With the advances in dense wavelength division multiplexing (DWDM) technology [21, 32, 38], the number of wavelengths in a wavelength division multiplexed (WDM) network increases to hundreds or more per fiber, and each wavelength operates at 10Gbps (OC-192) or higher [17, 18, 19]. While raw bandwidth has increased by more than four orders of magnitude over the last decade or so, capacity of switches has only been up by a factor of ten. Switching speed is the bottleneck at the core of optical network infrastructure [37]. Consequently, a challenge is to design cost-effective WDM cross-connects (WXC) that can scale in size beyond one hundred inputs and outputs and, at the same time, switch fast (e.g., tens of nanoseconds or less).

The notion of “cost-effectiveness” is difficult to capture. One can analyze and compare WDM switches both qualitatively and quantitatively.

Qualitatively, we need to know if a design is strictly nonblocking (SNB), rearrangeably nonblocking (RNB), and/or wide-sense nonblocking (WSNB) under different request models [20, 24, 25, 31, 33, 34, 39, 41, 42] and different traffic patterns (unicast [24, 25, 42], multicast [22, 26, 43]). A design can also be blocking as long as its blocking probability is below a certain threshold [15, 31]. There are various other qualitative features, such as small cross-talk [39], small number of limited-range wavelength converters [42], or fault tolerance [4]. Presumably each new design is guided by a particular qualitative feature. For example, one might come up with an RNB design under one request model, which may or may not be SNB under another request model. One might also have an intuitively good design, and hence need to know what qualitative feature the design possesses. This question is challenging in general. We will see later that the graph models introduced in this paper help, in several ways, answer these types of questions.

Quantitatively, comparing different designs, or asking how close to be optimal a new design is, are very important questions. This is a multidimensional problem, as there are many factors affecting the “cost” of a switch. Some factors, such as actual

*Received by the editors March 31, 2004; accepted for publication (in revised form) July 13, 2005; published electronically February 17, 2006.

<http://www.siam.org/journals/sicomp/35-3/44271.html>

[†]Department of Computer Science and Engineering, 201 Bell Hall, State University of New York at Buffalo, Amherst, NY 14260 (hungngo@cse.buffalo.edu).

cost in dollars, are business matters. Other factors include the numbers of different types of switching components, such as (de)multiplexors (MUXs/DEMUXs), full and limited wavelength converters (FWCs and LWCs), semiconductor optical amplifiers (SOAs), optical add-drop multiplexors (OADMs), directional couplers (DCs), etc; or signal and switch quality parameters, such as cross-talk, power consumption and attenuation, integratability and scalability, blocking probabilities, etc.

It should be apparent that we cannot hope to have a cost model that fits all needs. However, one can devise cost models which give good approximated measures on how “complex” a construction is. The notion of complexity should roughly capture as many practical parameters as possible.

In this paper, we outline an intriguing approach to model switch complexity which not only helps analyze WDM switches quantitatively and qualitatively but also suggests interesting generalizations of classical switching network theory [2, 29]. Then we address several important problems arising from the framework.

We consider two dominant request models in this paper. The following phenomena are samples of what our model suggests:

- (a) Designing WXC’s in the so-called $(\lambda, F, \lambda', F')$ -request model is basically the same as designing a circuit switch. Hence, many old ideas on circuit switching can be readily reused (section 4.1).
- (b) Two SNB switches in two models are equivalent topologically, even though one request model is much less restrictive than the other (section 5.1).
- (c) There is an inherent trade-off between a WXC’s “depth” (which is proportional to signal attenuation, cross-talk) and its “size” (which approximates the WXC’s complexity) (section 5.2).
- (d) Different designs of WXC’s which make use of different optical components can now be viewed in a unified manner. We can tell if two different-looking designs are equivalent topologically, for example (section 7).

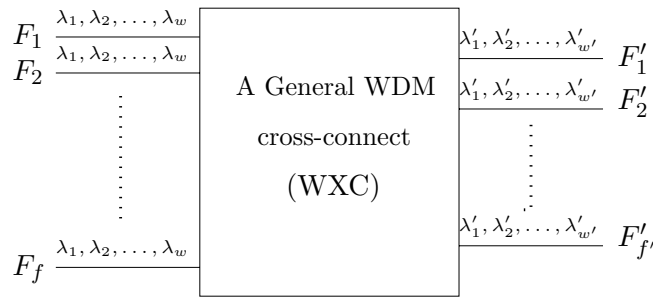
We will also derive several complexity bounds and give a generic construction which can be used to construct RNB switches (section 6).

The framework proposed here gives rise to interesting mathematical and networking problems, many of which are generalized versions of the well-studied circuit switching problems. We address several of these problems in later sections of the paper.

The rest of the paper is organized as follows. Section 2 introduces basic settings of WXC’s, request models, and nonblocking concepts. Section 3 motivates the graph models which will be rigorously defined in section 4. Section 5 addresses several key complexity problems arising from the framework. Section 6 explicitly constructs graphs with low complexity. The ideas in this section can be used to construct WXC’s of low cost. Section 7 discusses several applications of our framework. Last, section 8 concludes the paper with a few remarks and discussions on future works.

2. WXC’s, request models, and nonblockingness. A general WXC consists of f input fibers, each of which can carry a set $\Lambda = \{\lambda_1, \dots, \lambda_w\}$ of w wavelengths, and f' output fibers, each of which can carry a set $\Lambda' = \{\lambda'_1, \dots, \lambda'_{w'}\}$ of w' wavelengths, where $fw = f'w'$ (See Figure 1). This setting is referred to as the *heterogeneous* case [34], which is needed to connect subnetworks from different manufacturers. Henceforth, let $n = fw = f'w'$, unless specified otherwise.

Let $\mathcal{F} = \{F_1, \dots, F_f\}$ and $\mathcal{F}' = \{F'_1, \dots, F'_{f'}\}$ denote the set of input and output fibers, respectively. There are two common types of request models [24, 41]. In the (λ, F, F') -request model, a connection request is of the form (λ, F, F') , where $\lambda \in \Lambda$,

FIG. 1. *Heterogeneous WXC.*

$F \in \mathcal{F}$, and $F' \in \mathcal{F}'$. The request asks to establish a connection from wavelength λ in input fiber F to any free wavelength in output fiber F' . In the (λ, F, F') -request model, the difference is that the output wavelength λ' in F' is also specified.

In the next sections, we will define the concepts of SNB, WSNB, and RNB for both request models. We will be somewhat informal in our definitions. However, the idea should be clear to readers who have been exposed to switching theory [2, 13, 29, 23].

Consider a WXC with a few connections already established. Under the (λ, F, F') -model, a new request (λ, F, F') is said to be *valid* iff λ is a free wavelength in fiber F , and there are at most $w' - 1$ existing connections to F' . Under the $(\lambda, F, \lambda', F')$ -model, a new request $(\lambda, F, \lambda', F')$ is *valid* iff λ is free in F and λ' is free in F' .

A *request frame* under the (λ, F, F') -model is a set of requests such that no two requests are from the same wavelength in the same input fiber and that there are at most w' requests to any output fiber. A *request frame* under the $(\lambda, F, \lambda', F')$ -model is a set of requests such that no two requests are *from* the same input wavelength/fiber pair or *to* the same output wavelength/fiber pair.

The following definitions hold for both request models. A request frame is *realizable* by a WXC if all requests in the frame can be routed simultaneously. A WXC is *RNB* iff any request frame is realizable by the WXC. A WXC is *SNB* iff a new valid request can always be routed through the WXC without disturbing existing connections. A WXC is *WSNB* iff a new valid request can always be routed through the WXC without disturbing existing connections, provided that new requests are routed according to some routing algorithm. When the routing algorithm is known, we say that the WXC is WSNB with respect to the algorithm.

Henceforth, for any positive integer p , let $[p]$ denote the set $\{1, \dots, p\}$ and S_p denote the set of all permutations on $[p]$. The graph theoretic terminologies and notation we use here are fairly standard (see [40], for instance).

3. Motivations. Main known results on the constructions of (different types of) nonblocking WXCs can be found in [20, 22, 24, 25, 26, 31, 33, 34, 39, 41, 42]. (Note that we are not discussing multicast switching in this paper.) The constructions from these references made use of various different types of optical components, such as arrayed waveguide grating routers (AWGRs) and LWCs in [24], SOAs and LWCs in [42], OADMs and FWCs in [41], wavelength selective cross-connects (WSCs) and wavelength interchangers (WIs) in [33, 34], and DCs in [39]. It is clear that the task of comparing different designs is not easy. Different designs make use of different optical switching components which oftentimes are trade-offs. For instance, the designs in [42] made use of SOAs and LWCs which have a lower wavelength conversion cost than

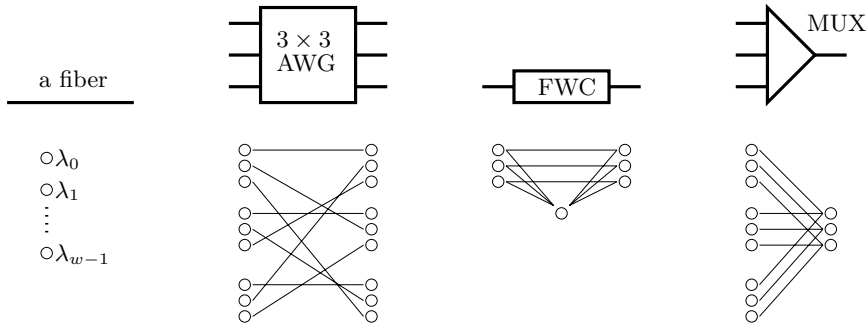


FIG. 2. Turning optical components into parts of a graph. A fiber is replaced by a set of vertices representing the wavelengths it can carry. Other components define edges connecting input wavelengths to output wavelengths. For the AWGR, MUX, and FWC, we illustrate with $w = 3$. Edges are directed from left to right.

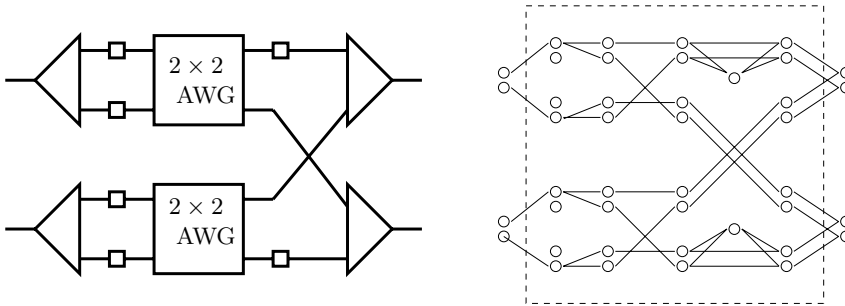


FIG. 3. A WDM switch design and its corresponding DAG.

those in [24]. On the other hand, the ones in [24] preferred AWGRs over SOAs since AWGRs consume virtually no power.

We now propose an approach to uniformly model all designs by graphs and then discuss switch complexity from the graphs' standpoint.

We classify optical switching components into fibers and other switching components. For any switch design, we apply the following procedure to construct a directed acyclic graph (DAG) from the design: (a) each fiber is replaced by a set of vertices $\Lambda \cup \Lambda'$, which represents all possible wavelengths which can be carried on the fiber; (b) the edges of the DAG are defined according to the capacity of switching components in the design. The edges connect wavelengths (i.e., vertices) on the inputs of each switching component to the wavelengths on the outputs in accordance with the functionality of the switching component.

We shall be somewhat brief on this construction. However, the reader will undoubtedly understand the basic idea. As an example, Figure 2 shows how to turn an AWGR, an FWC, and a MUX into edges. Figure 3 shows a complete construction of the DAG from the design on the left.

The key point is that a set of compatible routes from input wavelengths to output wavelengths correspond to a set of vertex disjoint paths from the inputs to the outputs of the DAG.

There are two main parameters of the DAG, which capture the notion of "switch complexity" discussed earlier. The number of edges of the DAG, called the size of the

DAG, is roughly proportional to the total cost of various components in the design. For example, an FWC corresponds to $3w$ edges, while an WI [34] corresponds to w^2 edges; a $w \times w$ AWGR corresponds to w^2 edges, while a $w \times w$ WDM crossbar corresponds to w^4 edges; etc. As WIs and WDM crossbars are more expensive than FWCs and AWGRs, this model makes sense. Other components follow the same trend.

The reader might have noticed that different components contribute different “weights” to the total cost; hence summing up the number of edges may not give the “right” cost. To answer this doubt, we make three points. First, as argued earlier one cannot hope to have a perfect model which fits all needs, and part of the notion of cost is a business matter. Our first aim is at a more theoretical level. Second, this is the first step toward a good cost model. One certainly can envision weighted graphs as the next step. Third, we surely can and should still use more traditional cost functions, such as the direct counts of the number of each components, and compare them individually.

The second measure on the DAG is its *depth*, i.e., the length of a longest path from any input to any output. As signals pass through different components of a design, they lose some power. The depth of the DAG hence reflects power loss or in some cases even the signal delay. Again, different components impose different power loss factors. Hence, other information needs to be taken into account to estimate power loss. However, it is clear that network depth is an important measure.

Last but not least, this DAG model provides a nice bridge between classical switching theory and WDM switching theory. As we shall see in later sections, this model helps us tremendously in answering qualitative questions about a particular construction. For example, if an wf -input wf -output DAG must have size $\Omega(f^2w^2)$ to be SNB, then we know for certain that a construction of cost $o(f^2w^2)$ (reflected by the DAG’s size) cannot be SNB (for sufficiently large values of fw .)

4. Rigorous settings. An (n_1, n_2) -network is a DAG $\mathcal{N} = (V, E; A, B)$, where V is the set of vertices, E is the set of edges, A is a set of n_1 nodes called *inputs*, and B , disjoint from A , is a set of n_2 nodes called *outputs*. The vertices in $V - A \cup B$ are *internal* vertices. The indegrees of the inputs and the outdegrees of the outputs are 0. The *size* of a network is its number of edges. The *depth* of a network is the maximum length of a path from an input to an output. An n -network is an (n, n) -network.

An n -network is meant to represent the DAG from the previous section under the $(\lambda, F, \lambda', F')$ -request model. (Recall $n = wf = w'f'$.) Later on, we shall define $[w, f]$ -networks which represent the DAG under the (λ, F, F') -request model.

4.1. The $(\lambda, F, \lambda', F')$ -request model. Given an n -network $\mathcal{N} = (V, E; A, B)$, a pair $D = (a, b)$ in $A \times B$ is called a *request* (or *demand*) for \mathcal{N} . A set \mathcal{D} of requests is called a *request frame* iff no two requests share an input or an output. A request $D = (a, b)$ is *compatible* with a request frame \mathcal{D} iff $\mathcal{D} \cup \{D\}$ is also a request frame. A *route* R for a request $D = (a, b)$ is a (directed) path from a to b . We also say R *realizes* D . A *state* of \mathcal{N} is a set \mathcal{R} of vertex disjoint routes. Each state of \mathcal{N} realizes a request frame, one route per request in the frame. A request frame \mathcal{D} is *realizable* iff there is a network state realizing it.

An *RNB n -connector* (or just n -connector for short) is an n -network in which the request frame $\mathcal{D} = \{(a, \pi(a)) \mid a \in A\}$ is realizable for any one-to-one correspondence $\pi : A \rightarrow B$.

An *SNB n -connector* is an n -network \mathcal{N} in which given any network state \mathcal{R} realizing a request set \mathcal{D} , and given a new request D compatible with \mathcal{D} , there exists a route R such that $\mathcal{R} \cup \{R\}$ is a network state realizing $\mathcal{D} \cup \{D\}$.

As requests come and go, a strategy to pick new routes for new requests is called a *routing algorithm*. An n -network \mathcal{N} is called a *WSNB n -connector* with respect to a routing algorithm \mathbf{A} if \mathbf{A} can always pick a new route for a new request compatible with the current network state. We can also replace \mathbf{A} by a class of algorithms \mathcal{A} . In general, an n -network \mathcal{N} is WSNB iff it is WSNB with respect to *some* algorithm.

We often consider two classes of functions on each network type: (a) the minimum size of a network and (b) the minimum size of a network with a given depth. The main theme of research on classical switching networks is to investigate the trade-off between size and depth [23, 29].

Let $rc(n)$, $wc(n)$, and $sc(n)$ denote the minimum size of an RNB, WSNB, and SNB n -connector, respectively. Let $rc(n, k)$, $wc(n, k)$, and $sc(n, k)$ denote the minimum size of an RNB, WSNB, and SNB n -connector with depth k , respectively. Note that $rc(n) \leq wc(n) \leq sc(n)$, and $rc(n, k) \leq wc(n, k) \leq sc(n, k)$. These classes of functions are well studied in the context of circuit switching networks (see, e.g., [23, 29] for nice surveys).

Two key conclusions arise from this formulation:

- Studying WDM switches under the $(\lambda, F, \lambda', F')$ -request model is in a sense the same as studying classical switching networks. A lot of results can be readily reused. For example, using our DAG construction, it is easy to see that all of the constructions (under this request model) in [24, 33, 34, 42] made use of various forms of the Clos network [5, 27], banyan, butterfly, and base line networks [11, 12], Cantor network [3], etc. In fact, under this request model, we do not know of any design which is not topologically isomorphic to some classical circuit design.
- The situation under the (λ, F, F') -model is different, however. The RNB design in [24] and the designs presented in this paper require several new themes. Particularly, this is because the (λ, F, F') -model is not equivalent to the classical switching case, as we shall see in the next section.

4.2. The (λ, F, F') -request model. In this request model, each pair (λ, F) with $\lambda \in \Lambda$, $F \in \mathcal{F}$ can still be thought of as an “input” to our graphs as in the previous request model. However, on the output side we do have to indicate the number f of fibers and the number of wavelengths w on each fiber.

Set $n = wf$. A $[w, f]$ -network is an n -network $\mathcal{N} = (V, E; A, B)$ in which the set B of outputs is further partitioned into f subsets B_1, \dots, B_f of size w each. Each set B_i represents an output fiber in the WDM switch. We implicitly assume the existence of the partition in a $[w, f]$ -network, in order to simplify notation. (There is a slightly subtle point to be noticed here. The inputs are not distinguishable in this request model, while we do care which fiber an output wavelength is from. The parameters w and f in the above sentence and henceforth should be thought of as w' and f' in the original discussion.)

Given a $[w, f]$ -network \mathcal{N} , a pair $D = (a, k) \in A \times [f]$ is called a (connection) *request* for \mathcal{N} . The number k is called the *output fiber number* of D . A set \mathcal{D} of requests is called a *request frame* iff no two requests share an input, and for any $k \in [f]$, we have $|\{a \mid (a, k) \in \mathcal{D}\}| \leq w$. A request $D = (a, k)$ is *compatible* with a request frame \mathcal{D} iff $\mathcal{D} \cup \{D\}$ is also a request frame.

A *route* R for a request $D = (a, k)$ is a path from a to some vertex b in B_k . We also say R *realizes* D . A *state* of \mathcal{N} is a set \mathcal{R} of vertex disjoint routes. Each state of \mathcal{N} realizes a request frame. A request frame \mathcal{D} is *realizable* iff there is a network state realizing it.

We are interested in (WSNB, SNB, RNB) connectors under this request model. A (rearrangeable) $[w, f]$ -connector is a $[w, f]$ -network in which the request frame

$$\mathcal{D} = \{(a, \sigma(a)) \mid a \in A\}$$

is realizable for any mapping $\sigma : A \rightarrow [f]$ such that

$$|\{a \mid \sigma(a) = k\}| = w \quad \forall k \in [f].$$

A SNB $[w, f]$ -connector is an $[w, f]$ -network \mathcal{N} in which given any network state \mathcal{R} realizing a request set \mathcal{D} , and given a new request D compatible with \mathcal{D} , there exists a route R such that $\mathcal{R} \cup \{R\}$ realizes $\mathcal{D} \cup \{D\}$. As requests come and go, a strategy to pick new routes for new requests is called a *routing algorithm*. An $[w, f]$ -network \mathcal{N} is called a WSNB $[w, f]$ -connector with respect to a routing algorithm \mathbf{A} if \mathbf{A} can always pick a new route for a new request compatible with the current network state. We can also replace \mathbf{A} by a class of algorithms \mathcal{A} . In general, an $[w, f]$ -network \mathcal{N} is WSNB iff it is WSNB with respect to *some* algorithm.

The different $[w, f]$ -networks are generalized versions of the corresponding n -networks.

PROPOSITION 4.1. *A network is an SNB, WSNB, RNB $[1, f]$ -connector iff it is an SNB, WSNB, RNB f -connector, respectively.*

Let $\overline{rc}(w, f)$, $\overline{wc}(w, f)$, and $\overline{sc}(w, f)$ denote the minimum sizes of an RNB, WSNB, and SNB $[w, f]$ -connector, respectively. Similarly, for a fixed depth k , we define $\overline{rc}(w, f, k)$, $\overline{wc}(w, f, k)$, and $\overline{sc}(w, f, k)$. These functions have not been studied before. Some trivial bounds can be summarized as follows.

PROPOSITION 4.2. *Let $n = wf$; then the following hold:*

- (i) $\overline{rc}(w, f) \leq \overline{wc}(w, f) \leq \overline{sc}(w, f)$.
- (ii) $\overline{rc}(w, f, k) \leq \overline{wc}(w, f, k) \leq \overline{sc}(w, f, k)$.
- (iii) *An RNB, WSNB, SNB n -connector is also an RNB, WSNB, SNB $[w, f]$ -connector, respectively. Consequently, $\overline{rc}(\cdot) \leq rc(\cdot)$, $\overline{wc}(\cdot) \leq wc(\cdot)$, and $\overline{sc}(\cdot) \leq sc(\cdot)$, where the dots on the left-hand sides can be replaced by (w, f) or (w, f, k) and the dots on the right-hand sides by (n) or (n, k) , correspondingly.*

5. Complexity bounds.

5.1. SNB $[w, f]$ -connectors. We study SNB $[w, f]$ -connectors in this section. For $f = 1$, it is easy to see that $\overline{sc}(w, 1, k) = w + k - 1$. We assume $f \geq 2$ from here on.

Intuitively, an optimal SNB $[w, f]$ -connector might have (strictly) smaller size than an optimal SNB wf -connector, since the $(\lambda, F, \lambda', F')$ -request model is more restrictive than the (λ, F, F') -request model. However, the following theorem shows a somewhat surprising result: we can do no better than an SNB wf -connector when $f \geq 2$. This theorem explains rigorously why the authors in [24] could not construct SNB designs under the (λ, F, F') -model with a lower cost than the ones under the other model.

THEOREM 5.1. *Let $n = wf$, where n, w, f are positive integers, and $f \geq 2$. An n -network $\mathcal{N} = (V, E; A, B)$ is an SNB n -connector iff it is an SNB $[w, f]$ -connector.*

Proof. An SNB n -connector is also an SNB $[w, f]$ -connector, no matter how the fiber partitioning is done. For the converse, let \mathcal{N} be an SNB $[w, f]$ -connector. Let $B = B_1 \cup \dots \cup B_f$ be the partition of B . (Recall, by definition, that $|B_i| = w \quad \forall i \in [f]$ and that $|A| = wf$.)

Consider a state \mathcal{R} of this network. We shall show that if a is a free input and b is a free output, then there exists a route R from a to b such that $\mathcal{R} \cup \{R\}$ is a network state.

Let X be the set of free inputs and Y the set of free outputs. Note that $a \in X$, $b \in Y$, and $|X| = |Y|$.

Suppose $b \in B_k$ for some $k \in [f]$. Without loss of generality, we assume that there is no free output in any B_j for $j \neq k$. This can be accomplished by creating as many requests of the form (x, j) as possible, where $x \in X - \{a\}$ and $j \neq k$, until there is no more free outputs at the B_j with $j \neq k$. Then let \mathcal{R} be the new network state (which satisfies all new requests and also contains the old network state). An (a, b) -route compatible with \mathcal{R} is certainly compatible with the old network state.

We can now assume $Y \subseteq B_k$. Create $|X|$ requests of the form (x, k) , one for each $x \in X$. Since \mathcal{N} is a $[w, f]$ -connector, there is a route R_x for each x in X satisfying the following: (i) R_x starts from x and ends at some vertex in B_k , and (ii) $\mathcal{R} \cup \{R_x \mid x \in X\}$ is a network state.

If R_a is an (a, b) -route, then we are done. Moreover, if $|X| = |Y| = 1$, then R_a must be an (a, b) -route. Consequently, we can assume the following:

- $|X| = |Y| \geq 2$.
- R_a goes from a to some vertex $y \in B_k - \{b\}$.
- There is some $x \in X - \{a\}$ such that R_x ends at b .
- There is some vertex $a' \notin X$ and a route

$$R' = (a', v_1, \dots, v_p, b') \in \mathcal{R}$$

which goes from a' to a vertex $b' \in B_j$, where $j \neq k$. (The route $R' \in \mathcal{R}$ exists since we assumed that the vertices in $B_j, j \neq k$, are all busy.)

To this end, let

$$\mathcal{R}' = \mathcal{R} \cup \{R_t \mid t \in X\} - \{R_a, R_x, R'\}.$$

We shall show that there exists an (a, b) -route R for which $\mathcal{R}' \cup \{R', R\}$ is a state. The route R is then the route we are looking for, because $\mathcal{R} \subseteq \mathcal{R}' \cup \{R'\}$.

We first claim that there exists an (x, y) -route R_{xy} compatible with \mathcal{R}' . Consider the state $\mathcal{R}' \cup \{R_a\}$. The request (a', k) is valid (i.e., compatible with the request frame realized by $\mathcal{R}' \cup \{R_a\}$), and b is the only free output in B_k ; hence, there is an (a', b) -route $R_{a'b}$ such that $\mathcal{R}' \cup \{R_a, R_{a'b}\}$ is a state. Now in the state $\mathcal{R}' \cup \{R_{a'b}\}$ the request (x, k) is valid, and y is the only free output in B_k . Hence, there is an (x, y) -route R_{xy} such that $\mathcal{R}' \cup \{R_{a'b}, R_{xy}\}$ is a state. Consequently, there is an (x, y) -route R_{xy} compatible with \mathcal{R}' as claimed.

Now consider two cases as follows.

Case 1. Among all the (x, y) -routes which are compatible with \mathcal{R}' , there is some R_{xy} which is also vertex disjoint from R' . Then in the state $\mathcal{R}' \cup \{R', R_{xy}\}$ the request (a, k) is valid, and b is the only free output in B_k . Hence, there is an (a, b) -path compatible with $\mathcal{R}' \cup \{R'\}$ as desired.

Case 2. Every (x, y) -route compatible with \mathcal{R}' intersects R' at some point. Let R_{xy} be such an (x, y) -route whose last intersection vertex on (v_1, \dots, v_p) has the largest index, say v_j , for some $j \in [p]$. Then R_{xy} is composed of two parts: the part from x to v_j and the part from v_j to y .

Let $R_{a'y}$ be an (a', y) -path consisting of the part (a', v_1, \dots, v_j) of R' and the (v_j, y) -part of R_{xy} . Then certainly $R_{a'y}$ is compatible with \mathcal{R}' . In the state $\mathcal{R}' \cup \{R_{a'y}\}$

the request (a, k) is valid, and b is the only free output in B_k . Hence, there is an (a, b) -path R_{ab} compatible with $\mathcal{R}' \cup \{R_{a'y}\}$.

If R_{ab} does not intersect R' , then we are done. Otherwise, R_{ab} must intersect R' at some $v_{j'}$ for which $j' > j$. Similar to the previous paragraph, we can form an (a', b) -path $R_{a'b}$ compatible with \mathcal{R}' consisting of $(a', v_1, \dots, v_{j'})$ and the part of R_{ab} from $v_{j'}$ to b . Now consider the state $\mathcal{R}' \cup \{R_{a'b}\}$ in which y is the only free vertex in B_k . The request (x, k) is valid; hence there is some (x, y) -path compatible with $\mathcal{R}' \cup \{R_{a'b}\}$. This (x, y) -path must then intersect R' (since we are in Case 2) at some vertex after $v_{j'}$ (for compatibility with $R_{a'b}$), contradicting our choice of R_{xy} earlier. \square

COROLLARY 5.2. *The following hold for $f \geq 2$:*

- (i) $\overline{sc}(w, f, 1) = w^2 f^2$.
- (ii) $\overline{sc}(w, f, k) = \Omega((wf)^{1+1/(k-1)})$ and $\overline{sc}(w, f, k) = O((wf)^{1+1/\lfloor \frac{k+1}{2} \rfloor})$.
- (iii) $\overline{sc}(w, f) = \Theta(wf \lg(wf))$.

Proof. Let $n = wf$; then $\overline{sc}(w, f, k) = sc(n, k)$ by Theorem 5.1. The first equality is obvious. The fact that $sc(n, k) = O(n^{1+1/\lfloor \frac{k+1}{2} \rfloor})$ can be seen from the constructions in [3, 5, 27]. The lower bound $\Omega((wf)^{1+1/(k-1)})$ was shown in [8]. The fact that $sc(n) = \Theta(n \lg n)$ can be found in [1, 36]. The reader is referred to the surveys [23, 29] for more details on what is known about these functions. \square

5.2. Rearrangeable $[w, f]$ -connectors. In this section, we first devise lower bounds for the optimal size of RNB $[w, f]$ -connectors and connectors of a fixed depth. The upper bounds follow from explicit constructions presented in section 6.

An idea of Pippenger [28] can be used to show the following theorem.

THEOREM 5.3. *Every rearrangeable $[w, f]$ -connector must have size at least*

$$\frac{45}{7} wf \log_6 f + O(f) - O(f \lg w).$$

In particular, $\overline{rc}(w, f) = \Omega(wf \lg f)$.

Proof. The proof is completely similar to that of Pippenger's theorem and thus will not be repeated here. The only difference is that the number of valid request frames is no longer $n!$, as in the case of an n -connector. In our case, the total number of different request frames for \mathcal{N} is the multinomial coefficient

$$(1) \quad \underbrace{\binom{wf}{w, \dots, w}}_{f \text{ times}} = \frac{(wf)!}{(w!)^f} \geq \frac{\sqrt{2\pi wf} (wf/e)^{wf}}{e^{\frac{f}{12w}} (2\pi w)^{f/2} (w/e)^{wf}} = (2\pi w)^{1-\frac{f}{2}} e^{-\frac{f}{12w}} f^{wf+\frac{1}{2}},$$

where the inequality follows from Stirling's approximation [35]. \square

The bound $\Omega(wf \lg f)$ implies that for $w \leq f$, $[w, f]$ -connectors must have size at least $\Omega(wf \lg(wf))$, which is asymptotically no better than a wf -connector. This confirms our intuition that for small values of w , $[w, f]$ -connectors are almost the same as wf -connectors.

Fortunately, in WDM networks it is often the case that $w \geq f$; i.e., the number of wavelengths per fiber (in the hundreds) is often much larger than the number of fibers (in the tens). The next section shows that we can construct $[w, f]$ -connectors that are asymptotically less expansive than all known constructions of wf -connectors.

We next give lower bounds for fixed depth $[w, f]$ -connectors.

THEOREM 5.4. *The optimal size of a depth-1 $[w, f]$ -connector is $wf(wf - w + 1)$, namely $\overline{rc}(w, f, 1) = wf(wf - w + 1)$.*

Proof. In the next section, we shall construct depth-1 $[w, f]$ -connectors of size $wf(wf - w + 1)$, which proves the upper bound $\bar{rc}(w, f, 1) \leq wf(wf - w + 1)$.

For the lower bound, let $\mathcal{N} = (A \cup B, E; A, B)$ be a depth-1 $[w, f]$ -connector. Then \mathcal{N} is a (directed) bipartite graph where $|A| = wf$, $|B| = wf$, and B has a partition into $B_1 \cup \dots \cup B_f$, such that $|B_i| = w \forall i$. The network \mathcal{N} is a $[w, f]$ -connector iff for every partition of A into A_1, \dots, A_f with $|A_i| = w \forall i$, there exist f complete matchings from each A_i to each B_i .

It follows that each vertex $b \in B$ must have a neighbor in every w -subset of A . Consequently, each vertex $b \in B$ must be of degree at least $|A| - w + 1$. Hence, the number of edges of \mathcal{N} is at least $|B|(|A| - w + 1) = wf(wf - w + 1)$. \square

For $k \geq 2$, we can use an idea by Pippenger and Yao [30] on n -shifters to find a lower bound for depth- k $[w, f]$ -connectors. The proof is similar and is left as an exercise.

THEOREM 5.5. *Let $k \geq 2$ be an integer; then a depth- k $[w, f]$ -connector must have size at least $kwf^{1+1/k}$. Specifically, $\bar{rc}(w, f, k) = \Omega(kwf^{1+1/k})$.*

Noting that the function $kwf^{1+1/k}$ is minimized at $k = \ln f$, we get the result $\bar{rc}(w, f) = \Omega(wf \lg f)$ from the previous theorem (with a worse constant than $45/7$).

COROLLARY 5.6. *For $k \geq 2$, $\bar{rc}_2(w, f) \geq ewf \ln f$, where e is the base of the natural log.*

6. Explicit constructions. For any network \mathcal{N} , let $A(\mathcal{N})$ and $B(\mathcal{N})$ denote the set of inputs and outputs of \mathcal{N} , respectively. For any $[w, f]$ -network \mathcal{N} , we shall always use $B_1(\mathcal{N}), \dots, B_f(\mathcal{N})$ to denote the partition of $B(\mathcal{N})$. An important fact to notice is that all presumably “theoretic” constructions presented in this section can easily be converted to practical constructions. We shall not elaborate on this point due to space limitation.

6.1. Atomic networks. Let $\mathcal{B}(x, y) = (A \cup B; E)$ denote the complete $x \times y$ directed bipartite graph; i.e., $|A| = x$, $|B| = y$, and $E = A \times B$. The (x, y) -network $\mathcal{B}(x, y)$ is called an (x, y) -crossbar. When $x = y$, we use the shorter notation $\mathcal{B}(x)$ and call it the x -crossbar. For any positive integer m , let $\mathcal{M}(m) = (A \cup B; E)$ denote a perfect matching of size m from A into B . (Therefore, $|A| = |B| = m$.) An (n, m) -concentrator is an (n, m) -network where $n \geq m$, such that for any subset S of m inputs there exists a set of m vertex disjoint paths connecting S to the outputs.

6.2. Union networks and optimal depth-1 connectors. Let $\mathcal{N}_1, \dots, \mathcal{N}_f$ be (wf, w) -networks with input sets A_1, \dots, A_f and output sets B_1, \dots, B_f , respectively. For each $i = 1, \dots, f - 1$, let $\phi_i : A_i \rightarrow A_{i+1}$ be some one-to-one mapping. A *left union* or \triangleleft -union of $\mathcal{N}_1, \dots, \mathcal{N}_f$ is a $[w, f]$ -network \mathcal{N} constructed by identifying each vertex $a \in A_1$ with all vertices $\phi_1(a), \phi_2 \circ \phi_1(a), \dots, \phi_{f-1} \circ \dots \circ \phi_1(a)$ (to become an input of \mathcal{N}), and let B_1, \dots, B_f be, naturally, the partition of the outputs of \mathcal{N} (see Figure 4.) We denote \mathcal{N} as $\mathcal{N} = \triangleleft(\mathcal{N}_1, \dots, \mathcal{N}_f)$.

Let $\mathcal{N}_1, \dots, \mathcal{N}_k$ be (m, n) -networks. An (mk, n) -network $\mathcal{N} = \triangleright(\mathcal{N}_1, \dots, \mathcal{N}_k)$ constructed by identifying outputs of the \mathcal{N}_i in some one-to-one manner is called a *right union* (or \triangleright -union) of the \mathcal{N}_i . The picture is virtually symmetrical to the left union picture.

The next theorem summarizes a few important properties of the union constructions. The proof is simple and thus omitted. Note that part (iii) completes the proof of Theorem 5.4.

THEOREM 6.1 (optimal depth-1 construction). *Let w, f be positive integers; then the following hold:*

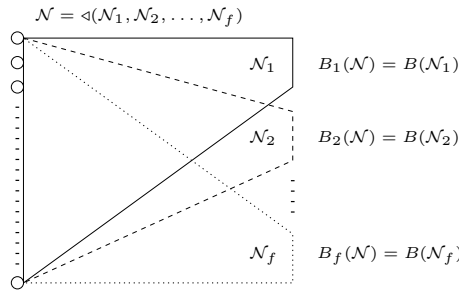


FIG. 4. The left union \mathcal{N} of f (wf, w) -networks is a $[w, f]$ -network.

- (i) Suppose $\mathcal{N}_1, \dots, \mathcal{N}_f$ are (wf, w) -concentrators; then the network $\triangleleft(\mathcal{N}_1, \dots, \mathcal{N}_f)$ is a $[w, f]$ -connector.
- (ii) The network $\mathcal{C}_1(w, f) = \triangleright(\mathcal{B}(wf - w, w), \mathcal{M}(w))$ is a depth-1 (wf, w) -concentrator of size $wf - w + 1$.
- (iii) Let $\mathcal{S}_1(w, f)$ be a left union of f copies of $\mathcal{C}_1(w, f)$. Then $\mathcal{S}_1(w, f)$ is a depth-1 $[w, f]$ -connector of size $wf(wf - w + 1)$, which is optimal.

6.3. Constructions of product networks and $[w, f]$ -connectors of depth 2.

DEFINITION 6.2 (the $\times \times$ -product). Let \mathcal{N}_1 be an m -network and \mathcal{N}_2 be a $[w, f]$ -network; then define the ordered product (for lack of a better term) $\mathcal{N} = \mathcal{N}_1 \times \times \mathcal{N}_2$ as follows. We shall “connect” wf copies of \mathcal{N}_1 , denoted by $\mathcal{N}_1^{(1)}, \dots, \mathcal{N}_1^{(wf)}$, to m copies $\mathcal{N}_2^{(1)}, \dots, \mathcal{N}_2^{(m)}$ of \mathcal{N}_2 . For each $i \in \{1, \dots, wf\}$ and $j \in \{1, \dots, m\}$, we identify the j th output of $\mathcal{N}_1^{(i)}$ with the i th input of $\mathcal{N}_2^{(j)}$. The output partition for \mathcal{N} is defined by

$$B_k(\mathcal{N}) = \bigcup_{j=1}^m B_k(\mathcal{N}_2^{(j)}).$$

Naturally, $A(\mathcal{N}) = \cup_{i=1}^{wf} A(\mathcal{N}_1^{(i)})$. Figure 5 illustrates the construction.

The following proposition summarizes a few trivial properties of the product network.

PROPOSITION 6.3. Let \mathcal{N}_1 be an m -network of size s_1 and depth d_1 and \mathcal{N}_2 a $[w, f]$ -network of size s_2 and depth d_2 . Then the network $\mathcal{N} = \mathcal{N}_1 \times \times \mathcal{N}_2$ is an $[mw, f]$ -network of size $s = wf s_1 + m s_2$ and depth $d = d_1 + d_2$.

Before proving a crucial property of this construction, we need a simple yet important lemma.

LEMMA 6.4. Let $G = (X \cup Y; E)$ be a bipartite multigraph where the degree of each vertex $x \in X$ is m and the degree of each vertex $y \in Y$ is mw . Then there is an edge coloring for G with exactly m colors such that vertices in X are incident to different colors, and vertices in Y are incident to exactly w edges of each color.

Proof. Split each vertex $y \in Y$ into w copies $y^{(1)}, \dots, y^{(w)}$ such that each copy has degree m . The resulting graph is an m -regular bipartite graph, which can be m -edge-colored, by König’s line coloring theorem [16]. This induces a coloring of G as desired. \square

The following lemma is the point of the ordered-product construction.

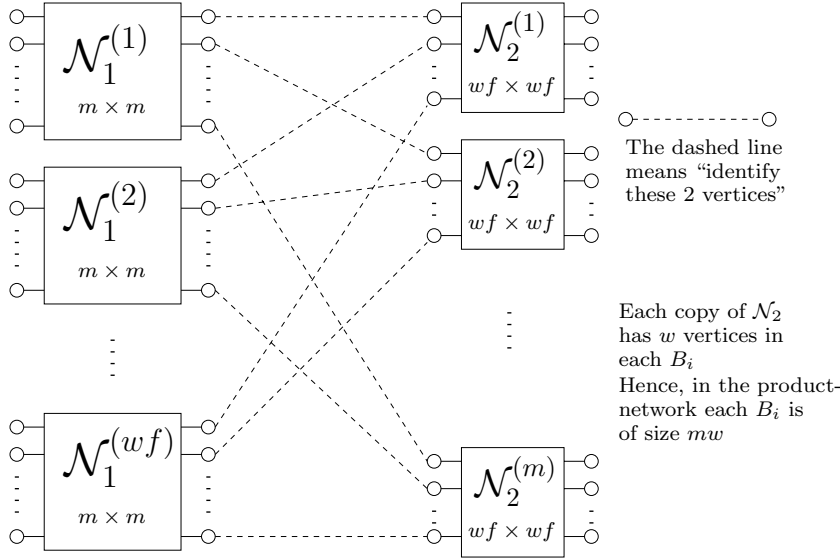


FIG. 5. Product of two networks: \mathcal{N}_1 is an m -network and \mathcal{N}_2 is a $[w, f]$ -network.

LEMMA 6.5. If \mathcal{N}_2 is a rearrangeable $[w, f]$ -connector and \mathcal{N}_1 is a rearrangeable m -connector, then $\mathcal{N} = \mathcal{N}_1 \times \times \mathcal{N}_2$ is a rearrangeable $[mw, f]$ -connector.

Proof. Consider a request frame \mathcal{D} for \mathcal{N} . We use $(a^{(i)}, k)$ to denote a request $(a, k) \in \mathcal{D}$ if $a \in A(\mathcal{N}_1^{(i)})$. This is to signify the fact that the request was from the a th input of $\mathcal{N}_1^{(i)}$ to B_k . By definition of a request frame, $|\{(a^{(i)}, k) \mid (a^{(i)}, k) \in \mathcal{D}\}| = m$ for a fixed i . We shall find vertex disjoint routes realizing requests in \mathcal{D} .

Construct a bipartite graph $G = (X \cup Y; E)$, where $X = \{\mathcal{N}_1^{(1)}, \dots, \mathcal{N}_1^{(wf)}\}$ is the set of all copies of \mathcal{N}_1 , and $Y = \{B_1, \dots, B_f\}$. There is (a copy of) an edge of G between $\mathcal{N}_1^{(i)}$ and B_k for each request $(a^{(i)}, k)$. Clearly, G is a bipartite graph satisfying the conditions of Lemma 6.4.

As each edge of G represents a request $D \in \mathcal{D}$, Lemma 6.4 implies that there is an m -coloring of all the requests such that, for a fixed i , requests of the form $(a^{(i)}, k)$ get different colors. Moreover, for a fixed k , requests of the form $(a^{(i)}, k)$ can be partitioned into m classes, where each class consists of exactly w requests of the same color.

Let $C = \{1, \dots, m\}$ be the set of colors. Let $c(a, k)$ denote the color of request $(a, k) \in \mathcal{D}$. Without loss of generality, we number the m outputs of $\mathcal{N}_1^{(i)}$ with numbers from 1 to m , i.e., $B(\mathcal{N}_1^{(i)}) = C, \forall i \in \{1, \dots, wf\}$.

Fix an $i \in \{1, \dots, wf\}$. As the m requests coming out of $\mathcal{N}_1^{(i)}$ have different colors, the correspondence $a^{(i)} \leftrightarrow c(a^{(i)}, k)$, where $(a^{(i)}, k) \in \mathcal{D}$, is a one-to-one correspondence between the inputs and the outputs of $\mathcal{N}_1^{(i)}$. Hence, for the m requests $(a^{(i)}, k)$, there exist m vertex disjoint routes $R_1(a^{(i)}, k)$ connecting input $a^{(i)}$ to the output numbered $c(a^{(i)}, k)$ of $\mathcal{N}_1^{(i)}$.

Fix a $j \in \{1, \dots, m\}$. The i th input of $\mathcal{N}_2^{(j)}$ is the j th output of $\mathcal{N}_1^{(i)}$, which is the endpoint of some route $R_1(a^{(i)}, k)$ for which $c(a^{(i)}, k) = j$. Let $k(i, j)$ be the number k such that the request $(a^{(i)}, k) \in \mathcal{D}$ has color $c(a^{(i)}, k) = j$. Then, for the fixed j and any $k \in \{1, \dots, f\}$, Lemma 6.4 ensures that there are exactly w of the $k(i, j)$ with

value k , namely $|\{i : k(i, j) = k\}| = w$. Thus, $\mathcal{D}' = \{(i, k(i, j)) \mid 1 \leq i \leq wf\}$ is a valid request frame for the rearrangeable $[w, f]$ -connector $\mathcal{N}_2^{(j)}$. Consequently, we can find vertex disjoint routes $R_2(i, k(i, j))$ connecting input i to some output in $B_{k(i, j)}$ of $\mathcal{N}_2^{(j)}$.

The concatenation of $R_1(a^{(i)}, k)$ and $R_2(i, k)$ completes a route realizing request $(a^{(i)}, k)$. These routes are vertex disjoint as desired. \square

We now illustrate the use of the Lemma 6.5 by a simple construction of depth-2 $[w, f]$ -connectors.

THEOREM 6.6 (depth-2 constructions). *Let w, f be positive integers; then*

- (i) *for $w \leq f - 1$, we can construct depth-2 $[w, f]$ -connectors of size $wf(w + f)$;*
- (ii) *for $w \geq f$, we can construct depth-2 $[w, f]$ -connectors of size $wf(2\sqrt{w(f - 1)} + 1)$.*

Proof. We ignore the issue of integrality for the sake of a clean presentation.

Write $w = mx$. By Theorem 6.1, $\mathcal{S}_1(x, f)$ is an $[x, f]$ -connector of depth 1 and size $xf(xf - x + 1)$. By Proposition 6.3 and Lemma 6.5, the network $\mathcal{B}(m) \times \times \mathcal{S}_1(x, f)$ is a $[w, f]$ -connector of depth-2 and size

$$s(x) = xfm^2 + mx f(xf - x + 1) = wf(w/x + (f - 1)x + 1).$$

Minimizing $s(x)$ as a function of x , with $1 \leq x \leq w$, we get the desired results. We pick $x = 1$ in case (i) and $x = \sqrt{w/(f - 1)}$ in case (ii). \square

6.4. Recursive constructions. Toward the constructions of $[w, f]$ -connectors, we need a few more definitions and properties.

DEFINITION 6.7. *Let w_0, w_1, \dots, w_k and f be positive integers and G be any $[w_0, f]$ -network. Let $\mathcal{N}(w_k, \dots, w_1; G)$ denote the recursively constructed network defined as follows:*

$$\begin{aligned} \mathcal{N}(\cdot; G) &= G, \\ \mathcal{N}(w_k, \dots, w_1; G) &= \mathcal{B}(w_k) \times \times \mathcal{N}(w_{k-1}, \dots, w_1; G). \end{aligned}$$

LEMMA 6.8. *Given positive integers w_0, w_1, \dots, w_k and f , let $w = \prod_{i=0}^k w_i$ and G be any $[w_0, f]$ -connector of size $s(G)$ and depth $d(G)$. Then the network $\mathcal{N} = \mathcal{N}(w_k, \dots, w_1; G)$ is a $[w, f]$ -connector of size*

$$s(\mathcal{N}) = w_0 \dots w_k f \cdot (w_1 + \dots + w_k) + w_1 \dots w_k \cdot s(G)$$

and depth $d(\mathcal{N}) = (k + d(G))$. (We set $s(\mathcal{N}) = s(G)$ when $k = 0$.)

Proof. This follows from Proposition 6.3 and Lemma 6.5. \square

PROPOSITION 6.9. *For any positive integers w and f , the following hold:*

- (i) *Let \mathcal{N} be any wf -connector. The $[w, f]$ -network \mathcal{N}' obtained by partitioning the outputs of \mathcal{N} arbitrarily into f subsets of size w is a $[w, f]$ -connector.*
- (ii) *$\mathcal{B}(f)$ is an f -connector and also a $[1, f]$ -connector.*

Basically, Proposition 6.9 implies that one can use good w_0f -networks to serve as the network G in Lemma 6.8. A general $[w, f]$ -network can then be constructed by decomposing $w = w_0, \dots, w_k$ with the right set of divisors w_0, \dots, w_k . As $[w, f]$ -networks of depth 2 have been constructed, we shall attempt to construct good networks of general depth and networks of a fixed depth at least 3.

Pippenger [27] has constructed a rearrangeable n -network, which we shall call $\mathcal{P}(n)$, of size $6n \log_3 n + O(n)$. He also constructed rearrangeable n -networks of depth $2i + 1$, $i \geq 1$, and size $2(i + 1)n \left(\frac{n}{2}\right)^{1/(i+1)} + O(n)$. An n -connector of depth $2i + 2$ can be

constructed by concatenating an n -matching with a depth- $(2i+1)$ n -connector. Hence, we can construct an n -connector of depth $j \geq 3$ and size $2\lceil j/2 \rceil n \left(\frac{n}{2}\right)^{1/\lceil j/2 \rceil} + O(n)$. We denote this network by $\mathcal{P}_j(n)$. For $j = 2$, a construction of size $O(n^{5/3})$ was given in [7] and [14]. Abusing notation, we shall also use $\mathcal{P}_2(n)$ to denote an n -connector of depth 2 and size $O(n^{5/3})$.

In the following results, we ignore the issue of integrality for the sake of clarity. We first address the general depth case.

THEOREM 6.10. *We can construct rearrangeable $[w, f]$ -connectors of size*

$$e \cdot wf \ln w + \frac{6}{\ln 3} wf \ln f + O(fw).$$

Proof. Let $w = xw_1 \dots w_k$. By Lemma 6.8 the network

$$\mathcal{N} = \mathcal{N}(w_k, \dots, w_1; \mathcal{P}(xf))$$

is a $[w, f]$ -connector of size

$$\begin{aligned} s(\mathcal{N}) &= wf(w_1 + \dots + w_k) + 6fx \log_3(fx) + O(fx) \\ &\geq wf \cdot k \cdot \left(\frac{w}{x}\right)^{1/k} + 6fx \log_3(fx) + O(fx). \end{aligned}$$

The right-hand side is minimized at $x = 1$ and $k = \ln w$. Equality can be obtained by setting $w_i = w^{1/k} \forall i$. \square

We now consider the fixed depth case. The networks $\mathcal{P}_j(n)$ are to be used. The following three theorems apply Lemma 6.8 with $G = \mathcal{P}_1, \mathcal{P}_2$, or \mathcal{P}_j with $j \geq 3$. Depending on the relative values between f, w , and k , one theorem may be better than the others.

THEOREM 6.11. *Let w, f , and $k \geq 3$ be positive integers.*

(i) *If $w < (f - 1)^{k-1}$, then we can construct a $[w, f]$ -connector of depth k and size*

$$(2) \quad (k - 1)fw^{1+\frac{1}{k-1}} + wf^2 = O(kwf^2).$$

(ii) *If $w \geq (f - 1)^{k-1}$, then we can construct a $[w, f]$ -connector of depth k and size*

$$(3) \quad wf(k - 1)[w(f - 1)]^{\frac{1}{k}} + w^{1+\frac{1}{k}}f(f - 1)^{\frac{1}{k}} + wf\Theta\left(k(wf)^{1+\frac{1}{k}}\right).$$

Proof. Write $w = xw_1 \dots w_{k-1}$. Then Lemma 6.8 implies that

$$\mathcal{N}(w_{k-1}, \dots, w_1; \mathcal{S}_1(x, f))$$

is a $[w, f]$ -network of depth k and size

$$(4) \quad \begin{aligned} s &= wf(w_1 + \dots + w_{k-1}) + w_1 \dots w_{k-1} \cdot xf(xf - x + 1) \\ &\geq wf(k - 1)(w/x)^{1/(k-1)} + wf(x(f - 1) + 1). \end{aligned}$$

Minimizing the right-hand side with respect to x , we get the desired results.

In case (i), equality can be obtained when $w_i = w^{1/(k-1)} \forall i$, and $x = 1$. In case (ii), equality can be obtained when $w_i = w^{1/(k-1)} \forall i$, and $x = \left(\frac{w}{(f-1)^{k-1}}\right)^{1/k}$. \square

We omit the proofs of the next two theorems due to the similarity to the above proof.

THEOREM 6.12. *Let w, f , and $k \geq 3$ be positive integers. Then there are positive real constants c_1, c_2 , and c_3 such that the following hold:*

(i) *If $w < c_1 f^{\frac{2}{3}(k-2)}$, then we can construct a $[w, f]$ -connector of depth k and size*

$$(5) \quad (k-2)f \cdot w^{1+\frac{1}{k-2}} + c_2 w f^{\frac{5}{3}} = O\left(k w f^{\frac{5}{3}}\right).$$

(ii) *If $w \geq c_1 f^{\frac{2}{3}(k-2)}$, then we can construct a $[w, f]$ -connector of depth k and size*

$$(6) \quad (k-2) \cdot (w f)^{1+\frac{1}{k-\frac{3}{2}}} + c_3 w^{1+\frac{1}{k-\frac{3}{2}}} f^{\frac{6k-7}{6k-9}} = O\left(k(w f)^{1+\frac{1}{k-\frac{3}{2}}}\right).$$

The following result can be improved by finer analysis. We give a somewhat “cleaner” version.

THEOREM 6.13. *Let w, f , and $k \geq 4$ be positive integers.*

(i) *If $w < f$, then we can construct a $[w, f]$ -connector of depth k and size*

$$(7) \quad O\left(k w^{1+\frac{1}{(k+1)}} f^{1+\frac{2}{(k+1)}}\right).$$

(ii) *If $w \geq f$, then we can construct a $[w, f]$ -connector of depth k and size*

$$(8) \quad O\left(k(w f)^{1+\frac{3}{2(k+1)}}\right).$$

7. Applications of our framework. In this section, we outline several practical applications coming from our theoretical formulation presented earlier. We will present only representative results. The reader should be able to see the main line of thoughts, nevertheless.

The applications fall into two main categories: (a) explicit constructions of WXCs and (b) complexity comparisons of known constructions.

7.1. Explicit constructions of WXC. The ideas for explicit constructions come from the physical realizations of atomic networks (section 6.1), the union of networks (section 6.2), the \times -product of networks (section 6.3), and the recursive construction (section 6.4).

There are several ways to physically realize an (x, y) -crossbar $\mathcal{B}(x, y)$. Figure 6(a) shows one possibility, and it is self-explanatory. The one thing to notice is that each fiber is aimed to carry one wavelength only. Another possibility to realize $\mathcal{B}(x, y)$ is to use a combination of an AWGR of dimension $\max\{x, y\}$ and the same number of LWCs as was done in [24]. The advantage of using AWGRs over SOAs is that AWGRs consume virtually no power.

Our second atomic component is a depth-1 $(w f, w)$ -concentrator $\mathcal{C}_1(w, f)$, which can be constructed by taking the left union of a $\mathcal{B}(w f - w, w)$ and a perfect matching $\mathcal{M}(w)$ as illustrated in Figure 6(b).

Given the aforementioned two atomic networks, we readily have a construction of a one-stage RNB WXC as shown in Figure 7. There is one column of tunable input—fixed output LWCs at the end to ensure no wavelength conflict. This one column of LWCs is needed in all realizations of the theoretical constructions shown in section 6.

This construction has a cost a little higher than that of Theorem 6.1 because of the LWCs, which are of total cost $w^2 f$. Asymptotically, however, $w^2 f \ll w f(w f - w + 1)$;

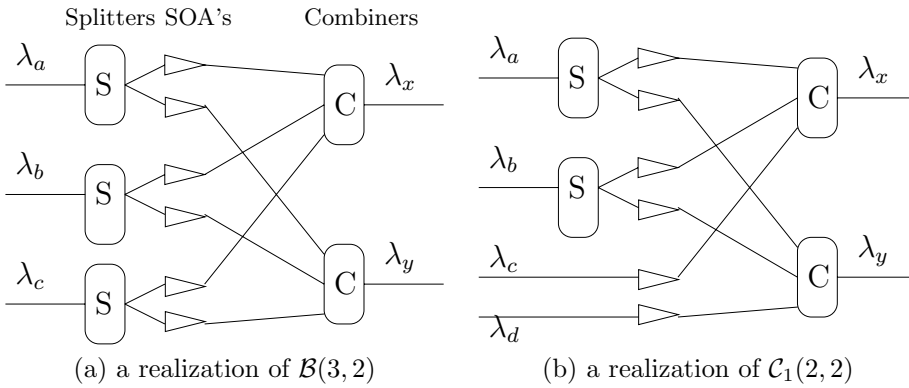


FIG. 6. Sample realizations of atomic networks.

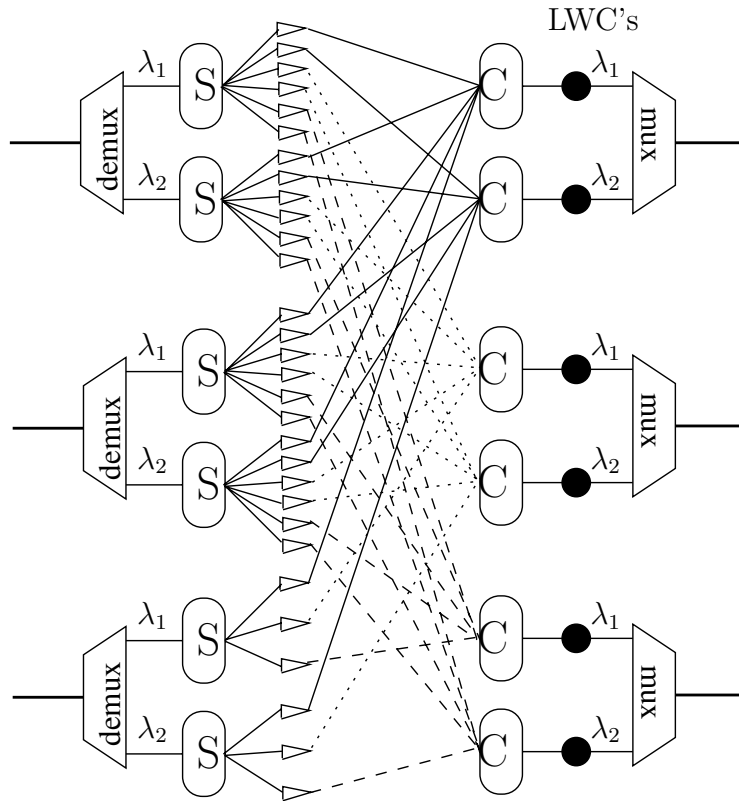


FIG. 7. A realization of the one-stage construction $\mathcal{S}_1(2, 3)$. The LWCs are wavelength converters with tunable inputs and a fixed output.

hence we did not create too much of a gap between the theoretical construction and the physical realization. Another interesting point to notice is that the WXC RNB-1 construction of [24] is a special case of this idea. The only difference is that they use AWGRs and LWCs to realize the crossbars.

The same idea can be used to realize the generic product network of Lemma 6.5. The proof of the lemma also gives an efficient routing algorithm. (Efficient algorithms

TABLE 1

Cost comparisons of different constructions under the two request models. SNB/RNB-1 refers to the (λ, F, F') -request model. SNB/RNB-2 refers to the $(\lambda, F, \lambda', F')$ -request model.

	Type	Depth	Size	Cond
[41]/CBC	SNB-1	$2 \lg f$	$\Theta(wf(w + \lg f) \lg f)$	-
Corollary 5.2(ii)	SNB-1	k	$O((wf)^{1+1/\lfloor \frac{k+1}{2} \rfloor})$	-
Corollary 5.2(iii)	SNB-1	$\lg(wf)$	$\Theta(wf \lg(wf))$	-
[41]/WI-Cantor	RNB-1	$2 \lg f$	$\Theta(wf(w + \lg^2 f))$	-
[41]/WI-Beneš	RNB-1	$2 \lg f$	$\Theta(wf(w + \lg f))$	-
[24]/WXC-RNB-1	RNB-1	2	$2(wf)^{3/2}$	$f < w$
Theorem 6.6(i)	RNB-1	2	$wf(w + f)$	$f > w$
Theorem 6.6(ii)	RNB-1	2	$2(wf)^{3/2}$	$f \leq w$
Theorem 6.10	RNB-1	$\lg(wf)$	$ewf \ln w + \frac{6}{\ln 3} wf \ln f + O(wf)$	-
Theorem 6.11	RNB-1	$k \geq 3$	$(k-1)fw^{1+\frac{1}{k-1}} + wf^2$	$w < (f-1)^{k-1}$
Theorem 6.11	RNB-1	$k \geq 3$	$\Theta\left(k(wf)^{1+\frac{1}{k}}\right)$	$w \geq (f-1)^{k-1}$
Theorem 6.12	RNB-1	$k \geq 3$	$(k-2)f \cdot w^{1+\frac{1}{k-2}} + c_2 wf^{5/3}$	$w < c_1 f^{\frac{2}{3}(k-2)}$
Theorem 6.12	RNB-1	$k \geq 3$	$O\left(k(wf)^{1+\frac{1}{k-3/2}}\right)$	$w \geq c_1 f^{\frac{2}{3}(k-2)}$
Theorem 6.13	RNB-1	$k \geq 4$	$O\left(kw^{1+\frac{1}{k+1}} f^{1+\frac{2}{k+1}}\right)$	$w < f$
Theorem 6.13	RNB-1	$k \geq 4$	$O\left(k(wf)^{1+\frac{3}{2(k+1)}}\right)$	$w < f$
[33, 34]	SNB-2	3	$2w^2(2f + w)$	-
[24]/WXC-SNB-2	SNB-2	3	$4\sqrt{2}(wf)^{3/2}$	$f < w$
[42]/2S/P/N	SNB-2	3	$4(wf)^{3/2}$	$f < w$
[42]/3S/P/N	SNB-2	4	$4\sqrt{2}(wf)^{3/2}$	-
[24]/WXC-RNB-2	RNB-2	3	$2(wf)^{3/2}$	$f < w$
[42]/2S/P/R	RNB-2	3	$2(wf)^{3/2}$	$f < w$
[42]/3S/P/R	RNB-2	4	$2\sqrt{2}(wf)^{3/2}$	-
[42]/B/P/R	RNB-2	$\Theta(\lg(wf))$	$\Theta(wf \lg wf)$	$f < w$

for bipartite graph edge coloring can be found in [6, 9, 10].) Thus, we readily have constructions of a depth-2 RNB WXC as in Theorem 6.6 and several different recursive constructions as reported in Theorems 6.10, 6.11, 6.12, and 6.13. Depending on the relationship between w and f , we pick the best one to use.

7.2. Complexity comparisons of known constructions. Table 1 summarizes the costs of various recent constructions (including some of the ones in this paper). The costs are assessed in terms of architectural depths and sizes. From the table, we see the following:

- For the $(\lambda, F, \lambda', F')$ -request model (SNB-1, RNB-1), the various constructions have costs asymptotically the same as those of their circuit switching counterparts. This was expected, since our formulation in section 4.1 has indicated that the WXC under this request model and the corresponding circuit switches are equivalent topologically.

(This is not to say that there is nothing to study in this request model. Our cost model does not capture precisely more practical criteria such as cross-talks, attenuation, and wavelength conversion costs.)

- For the (λ, F, F') -request model (SNB-1, RNB-1), there is much room for improvement.

In the SNB case, our construction of Corollary 5.2 is already better than the existing construction CBC. However, since SNB in this request model is the same as the other, we cannot expect much more improvement.

In the RNB case, our constructions of Theorems 6.10, 6.11, 6.12, and 6.13 are better than all known constructions. However, there are still gaps between the constructions and theoretical lower bounds of Theorems 5.3 and 5.5. We expect that both the lower bounds and the constructions can be improved until they are asymptotically equal.

8. Conclusions and future works. There are several benefits of the proposed graph models: they help analyze the switches qualitatively and quantitatively, they can be used to compare switch complexity, and they give rise to interesting mathematical problems relating to many areas, such as classical switching theory, graph theory, and algebraic graph theory. Some of these points were not discussed in the paper. It would be interesting, for instance, to investigate the use of expanders for constructing $[w, f]$ -connectors.

We have addressed several important problems arising from this framework, including studying optimal networks and their constructions, the trade-off between network depth and size, and the equivalence of networks under different request models. Some practical applications have also been pointed out.

Many problems remain open. In particular, we have not touched upon the WSNB case much. The multicast switch complexity was not considered. The asymptotic bounds of various complexity functions are still not optimal.

REFERENCES

- [1] L. A. BASSALYGO AND M. S. PINSKER, *The complexity of an optimal nonblocking commutation scheme without reorganization*, Problemy Peredači Informacii, 9 (1973), pp. 84–87.
- [2] V. E. BENEŠ, *Mathematical Theory of Connecting Networks and Telephone Traffic*, Math. Sci. Engrg. 17, Academic Press, New York, 1965.
- [3] D. G. CANTOR, *On nonblocking switching networks*, Networks, 1 (1971/72), pp. 367–377.
- [4] F. CAO AND D.-Z. DU, *Fault-tolerant routing and multicasting in butterfly networks*, in Proceedings of the 1999 ACM Symposium on Applied Computing (SAC '99, San Antonio, TX), 1999, pp. 455–460.
- [5] C. CLOS, *A study of nonblocking switching networks*, Bell System Tech. J., 32 (1953), pp. 406–424.
- [6] R. COLE AND J. HOPCROFT, *On edge coloring bipartite graphs*, SIAM J. Comput., 11 (1982), pp. 540–546.
- [7] P. FELDMAN, J. FRIEDMAN, AND N. PIPPENGER, *Wide-sense nonblocking networks*, SIAM J. Discrete Math., 1 (1988), pp. 158–173.
- [8] J. FRIEDMAN, *A lower bound on strictly nonblocking networks*, Combinatorica, 8 (1988), pp. 185–188.
- [9] H. N. GABOW, *Using Euler partitions to edge color bipartite multigraphs*, Internat. J. Comput. Information Sci., 5 (1976), pp. 345–355.
- [10] H. N. GABOW AND O. KARIV, *Algorithms for edge coloring bipartite graphs and multigraphs*, SIAM J. Comput., 11 (1982), pp. 117–129.
- [11] H. HINTON, *An Introduction to Photonic Switching Fabrics*, Plenum Press, New York, 1993.
- [12] J. H. HUI, *Switching and Traffic Theory for Integrated Broadband Networks*, Kluwer Academic Publishers, Boston, Dordrecht, London, 1990.

- [13] F. K. HWANG, *The Mathematical Theory of Nonblocking Switching Networks*, World Scientific, River Edge, NJ, 1998.
- [14] F. K. HWANG AND G. W. RICHARDS, *A two-stage network with dual partial concentrators*, *Networks*, 23 (1993), pp. 53–58.
- [15] X. JIANG, H. SHEN, M. UR RASHID KHANDKER, AND S. HORIGUCHI, *Blocking behaviors of crosstalk-free optical Banyan networks on vertical stacking*, *IEEE/ACM Transactions on Networking*, 11 (2003), pp. 982–993.
- [16] D. KÖNIG, *Über graphen und ihre anwendung auf determinantentheorie und mengenlehre*, *Math. Ann.*, 77 (1916), pp. 453–465.
- [17] LUCENT TECHNOLOGIES PRESS RELEASE, *Lucent Technologies Unveils Ultra-High-Capacity Optical System; Time Warner Telecom First to Announce it Will Deploy the System*, 2001, <http://www.lucent.com/press/0101/010117.nsa.html>.
- [18] LUCENT TECHNOLOGIES PRESS RELEASE, *Lucent Technologies Engineers and Scientists Set New Fiber Optic Transmission Record*, 2002, <http://www.lucent.com/press/0302/020322.bla.html>.
- [19] LUCENT TECHNOLOGIES WEBSITE, *What is Dense Wave Division Multiplexing (DWDM)?*, 2002, <http://www.bell-labs.com/technology/lightwave/dwdm.html>.
- [20] G. MAIER AND A. PATTAVINA, *Design of photonic rearrangeable networks with zero first-order switching-element-crosstalk*, *IEEE Trans. Comm.*, 49 (2001), pp. 1248–1279.
- [21] B. MUKHERJEE, *Optical Communication Networks*, McGraw-Hill, New York, 1997.
- [22] H. Q. NGO, *Multiwavelength distribution networks*, in *Proceedings of the 2004 Workshop on High Performance Switching and Routing (HPSR 2004, Phoenix, AZ)*, IEEE, Los Alamitos, CA, 2004, pp. 186–190.
- [23] H. Q. NGO AND D.-Z. DU, *Notes on the complexity of switching networks*, in *Advances in Switching Networks*, D.-Z. Du and Hung Q. Ngo, eds., Kluwer Academic Publishers, Dordrecht, The Netherlands, 2001, pp. 307–367.
- [24] H. Q. NGO, D. PAN, AND C. QIAO, *Nonblocking WDM switches based on arrayed waveguide grating and limited wavelength conversion*, in *Proceedings of the 23rd Conference of the IEEE Communications Society (INFOCOM'2004, Hong Kong)*, 2004.
- [25] H. Q. NGO, D. PAN, AND Y. YANG, *Optical switching networks with minimum number of limited range wavelength converters*, in *Proceedings of the 24rd Conference of the IEEE Communications Society (INFOCOM'2005, Miami, FL)*, 2005.
- [26] D. PAN, V. ANAND, AND H. Q. NGO, *Cost-effective constructions for nonblocking wdm multicast switching networks*, in *Proceedings of the 2004 International Conference on Communications (ICC'2004, Paris, France)*, IEEE, Los Alamitos, CA, 2004.
- [27] N. PIPPENGER, *On rearrangeable and nonblocking switching networks*, *J. Comput. System Sci.*, 17 (1978), pp. 145–162.
- [28] N. PIPPENGER, *A new lower bound for the number of switches in rearrangeable networks*, *SIAM J. Algebraic Discrete Methods*, 1 (1980), pp. 164–167.
- [29] N. PIPPENGER, *Communication networks*, in *Handbook of Theoretical Computer Science*, Vol. A, Elsevier, Amsterdam, 1990, pp. 805–833.
- [30] N. PIPPENGER AND A. C. C. YAO, *Rearrangeable networks with limited depth*, *SIAM J. Algebraic Discrete Methods*, 3 (1982), pp. 411–417.
- [31] J. RAMAMIRTHAM AND J. S. TURNER, *Design of wavelength converting switches for optical burst switching*, in *Proceedings of the 21st Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM)*, Vol. 2, 2002, pp. 1162–1171.
- [32] R. RAMASWAMI AND K. SIVARAJAN, *Optical Networks: A Practical Perspective*, 2nd ed., Morgan Kaufmann, San Francisco, CA, 2001.
- [33] A. RASALA AND G. WILFONG, *Strictly nonblocking WDM cross-connects*, in *Proceedings of the Eleventh Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'2000, San Francisco, CA)*, 2000, pp. 606–615.
- [34] A. RASALA AND G. WILFONG, *Strictly nonblocking WDM cross-connects for heterogeneous networks*, in *Proceedings of the 32nd Annual ACM Symposium on Theory of Computing (STOC'2000, Portland, OR)*, 2000, pp. 513–524.
- [35] H. ROBBINS, *A remark on Stirling's formula*, *Amer. Math. Monthly*, 62 (1955), pp. 26–29.
- [36] C. E. SHANNON, *Memory requirements in a telephone exchange*, *Bell System Tech. J.*, 29 (1950), pp. 343–349.
- [37] A. SINGHAL AND R. JAIN, *Terabit switching: A survey of techniques and current products*, *Computer Communications*, 25 (2002), pp. 547–556.
- [38] T. E. STERN AND K. BALA, *Multiwavelength Optical Networks: A Layered Approach*, Prentice-Hall, Upper Saddle River, NJ, 1999.
- [39] M. M. VAEZ AND C.-T. LEA, *Strictly nonblocking directional-coupler-based switching networks under crosstalk constraint*, *IEEE Trans. Comm.*, 48 (2000), pp. 316–323.

- [40] D. B. WEST, *Introduction to Graph Theory*, Prentice–Hall, Upper Saddle River, NJ, 1996.
- [41] G. WILFONG, B. MIKKELSEN, C. DOERR, AND M. ZIRNGIBL, *WDM cross-connect architectures with reduced complexity*, *Journal of Lightwave Technology*, 17 (1999), pp. 1732–1741.
- [42] Y. YANG AND J. WANG, *Designing WDM optical interconnects with full connectivity by using limited wavelength conversion*, *IEEE Trans. Comput.*, 53 (2004), pp. 1547–1556.
- [43] Y. YANG, J. WANG, AND C. QIAO, *Nonblocking WDM multicast switching networks*, *IEEE Trans. Parallel Distrib. Syst.*, 11 (2000), pp. 1274–1287.

**SPECIAL ISSUE: THE THIRTY-SIXTH ANNUAL ACM
SYMPOSIUM ON THEORY OF COMPUTING (STOC 2004)**

This volume comprises the polished and fully refereed versions of a selection of papers presented at the Thirty-Sixth Annual ACM Symposium on Theory of Computing (STOC 2004), held in Chicago, Illinois, June 13–15, 2004. Unrefereed preliminary versions of the papers presented at the symposium appeared in the proceedings of the meeting, published by ACM. The symposium was sponsored by the ACM Special Interest Group on Algorithms and Computation Theory (SIGACT).

The STOC 2004 Program Committee consisted of Andris Ambainis, László Babai (chair), Boaz Barak, Moses Charikar, Irit Dinur, Herbert Edelsbrunner, Sandy Irani, Adam Klivans, Vladlen Koltun, Robert Krauthgamer, Satya Lokam, Tal Malkin, Oded Regev, Alexander Russell, Éva Tardos, Mikkel Thorup, D. Sivakumar, Chris Umans, and Eric Vigoda.

Out of 271 “Extended Abstracts” submitted to the STOC 2004 Program Committee, 70 were selected for presentation at the symposium. Sixteen out of those 70 papers were selected for inclusion in this volume. The authors of 5 of the invited papers declined, and the remaining 11 accepted the invitation. One of the 11 papers was not completed by the deadline; one other paper was found to fall short of *SICOMP* standards. The present volume includes the remaining 9 papers.

This collection of papers encompasses a wide variety of questions and methods in theoretical computer science, often shedding new light on entire areas with a fresh approach. The topics include fundamental questions of complexity theory and algorithms as well as foundational mathematical problems. Several papers use methods of “continuous mathematics” to attack discrete optimization problems. Of the areas represented in this volume that have relatively recently gained prominence in the theory of computing, I should mention quantum computing and the theory of metric embeddings.

This issue includes the journal versions of the two papers that shared the 2004 Danny Lewin Best Student Paper award. One of them, by Scott Aaronson, shows how quantum arguments inspire new results in a classical model; the other, by Jonathan Kelner, demonstrates the relevance of conformal geometry to the algorithmic question of partitioning graphs into clusters.

All papers were refereed in accordance with *SICOMP*’s stringent standards, and most of them were substantially updated in the process. We take this opportunity to thank all the referees whose anonymous work has significantly contributed to the value of this volume.

* * * * *

Special issues dedicated to STOC have a distinguished history; a brief review of this history seems to be in order.

SIGACT, the premier U.S.-based organization of theoretical computer science, has sponsored the publication of special issues to its annual STOC conferences since the second STOC held in 1970; the *Journal of Computer and System Sciences (JCSS)* was designated the venue of the publication. *JCSS* published 34 STOC special issues, starting with the 2nd STOC (*JCSS* 5:3, June 1971) and ending with the 35th STOC (*JCSS* 69:3, November 2004). The volumes from 1978 onward are accessible to subscribers on the Elsevier website at <http://www.sciencedirect.com/>.

It was an honor to edit the present special issue for the *SIAM Journal on Computing*.¹

László Babai
Guest Editor
University of Chicago

¹My personal remarks on the change of venue can be found on my home page.

APPROXIMATING THE CUT-NORM VIA GROTHENDIECK'S INEQUALITY*

NOGA ALON[†] AND ASSAF NAOR[‡]

Abstract. The *cut-norm* $\|A\|_C$ of a real matrix $A = (a_{ij})_{i \in R, j \in S}$ is the maximum, over all $I \subset R$, $J \subset S$, of the quantity $|\sum_{i \in I, j \in J} a_{ij}|$. This concept plays a major role in the design of efficient approximation algorithms for dense graph and matrix problems. Here we show that the problem of approximating the cut-norm of a given real matrix is MAX SNP hard, and we provide an efficient approximation algorithm. This algorithm finds, for a given matrix $A = (a_{ij})_{i \in R, j \in S}$, two subsets $I \subset R$ and $J \subset S$, such that $|\sum_{i \in I, j \in J} a_{ij}| \geq \rho \|A\|_C$, where $\rho > 0$ is an absolute constant satisfying $\rho > 0.56$. The algorithm combines semidefinite programming with a rounding technique based on Grothendieck's inequality. We present three known proofs of Grothendieck's inequality, with the necessary modifications which emphasize their algorithmic aspects. These proofs contain rounding techniques which go beyond the random hyperplane rounding of Goemans and Williamson [*J. ACM*, 42 (1995), pp. 1115–1145], allowing us to transfer various algorithms for dense graph and matrix problems to the sparse case.

Key words. cut-norm, Grothendieck's inequality, semidefinite programming, approximation algorithms, Szemerédi partitions

AMS subject classifications. 68W25, 68Q01, 90C22, 90C27, 15A23

DOI. 10.1137/S0097539704441629

1. Introduction. The *cut-norm* $\|A\|_C$ of a real matrix $A = (a_{ij})_{i \in R, j \in S}$ with a set of rows indexed by R and a set of columns indexed by S is the maximum, over all $I \subset R$, $J \subset S$, of the quantity $|\sum_{i \in I, j \in J} a_{ij}|$. This concept plays a major role in the work of Frieze and Kannan [8] on efficient approximation algorithms for dense graph and matrix problems (see also [2] and the references therein). The techniques in [8] enabled the authors to approximate efficiently the cut-norm of an n by m matrix with entries in $[-1, 1]$ up to an *additive* error of ϵnm . However, prior to the present paper no polynomial time algorithm was known for approximating the cut-norm of a general real matrix within a constant multiplicative factor.

Let CUT NORM denote the computational problem of computing the cut-norm of a given real matrix. Here we first observe that the CUT NORM problem is MAX SNP hard, and then provide an efficient approximation algorithm for the problem. This algorithm finds, for a given matrix $A = (a_{ij})_{i \in R, j \in S}$, two subsets $I \subset R$ and $J \subset S$, such that $|\sum_{i \in I, j \in J} a_{ij}| \geq \rho \|A\|_C$, where $\rho > 0$ is an absolute constant. We first describe a deterministic algorithm that supplies a rather poor value of ρ , and then describe a randomized algorithm that provides a solution of expected value greater than 0.56 times the optimum.

The algorithm combines semidefinite programming with a rounding technique based on (the proofs of) Grothendieck's inequality. This inequality, first proved in [11],

*Received by the editors March 3, 2004; accepted for publication (in revised form) July 23, 2005; published electronically February 21, 2006. A preliminary version of this paper appeared in *Proceedings of the 36th Annual ACM Symposium on Theory of Computing (STOC)*, 2004.

<http://www.siam.org/journals/sicomp/35-4/44162.html>

[†]Schools of Mathematics and Computer Science, Raymond and Beverly Sackler Faculty of Exact Sciences, Tel Aviv University, Tel Aviv 69978, Israel (noga@math.tau.ac.il). This author's research was supported in part by a USA-Israeli BSF grant, by the Israel Science Foundation, and by the Hermann Minkowski Minerva Center for Geometry at Tel Aviv University.

[‡]Microsoft Research, One Microsoft Way 113/2131, Redmond, WA 98052-6399 (anaor@microsoft.com).

is a fundamental tool in functional analysis and has several interesting applications in this area. We will actually use the matrix version of Grothendieck's inequality, formulated in [20]. In order to apply semidefinite programming to the study of the cut-norm of an n by m matrix $A = (a_{ij})$, it is convenient to first study another norm,

$$\|A\|_{\infty \rightarrow 1} = \max \sum_{i=1}^n \sum_{j=1}^m a_{ij} x_i y_j,$$

where the maximum is taken over all $x_i, y_j \in \{-1, 1\}$.

It is not difficult to show (see section 3) that for every matrix A ,

$$4\|A\|_C \geq \|A\|_{\infty \rightarrow 1} \geq \|A\|_C,$$

and hence a constant approximation of any of these norms provides a constant approximation of the other.

The value of $\|A\|_{\infty \rightarrow 1}$ is given by the following quadratic integer program:

$$(1.1) \quad \begin{aligned} & \text{Maximize} && \sum_{ij} a_{ij} x_i y_j \\ & \text{subject to} && x_i, y_j \in \{-1, 1\} \quad \text{for all } i, j. \end{aligned}$$

The obvious semidefinite relaxation of this program is

$$(1.2) \quad \begin{aligned} & \text{Maximize} && \sum_{ij} a_{ij} u_i \cdot v_j \\ & \text{subject to} && \|u_i\| = \|v_j\| = 1, \end{aligned}$$

where here $u_i \cdot v_j$ denotes the inner product of u_i and v_j , which are now vectors of (Euclidean) norm 1 that lie in an arbitrary Hilbert space. Clearly we may assume, without loss of generality, that they lie in an $(n+m)$ -dimensional space.

This semidefinite program can be solved, using well-known techniques (see [10]) within an additive error of ϵ , in polynomial time (in the length of the input and in the logarithm of $1/\epsilon$). The main problem is the task of rounding this solution into an integral one. A first possible attempt is to imitate the technique of Goemans and Williamson in [12]; that is, given a solution u_i, v_j to the above program, pick a random vector z and define $x_i = \text{sign}(u_i \cdot z)$ and $y_j = \text{sign}(v_j \cdot z)$. It is easy to check that the expected value of $x_i y_j$ satisfies $E(x_i y_j) = \frac{2}{\pi} \arcsin(u_i \cdot v_j)$, and as $\arcsin(t)$ and t differ only in constant factors for all $-1 \leq t \leq 1$, one could hope that this will provide an integral solution whose value is at least some absolute constant fraction of the value of the optimal solution. This reasoning is, unfortunately, incorrect, as some of the entries a_{ij} may be positive and some may be negative. (In fact, the problem is interesting only if this is the case, since otherwise either $x_i = y_j = 1$ or $x_i = -y_j = 1$ for all i, j supplies the required maximum.) Therefore, even if each single term $a_{ij} u_i \cdot v_j$ is approximated well by its integral rounding $a_{ij} x_i y_j$, there is no reason to expect the sum to be well approximated, due to cancellations. We thus have to compare the value of the rounded solution to that of the semidefinite program on a global basis. Nesterov [22] obtained a result of this form for the problem of approximating the maximum value of a quadratic form $\sum_{ij} b_{ij} x_i x_j$, where $x_i \in \{-1, 1\}$, but only for the special case in which the matrix $B = (b_{ij})$ is positive semidefinite. While his estimate is global, his rounding is the same simple rounding technique of [12] described above.

As explained before, some new ideas are required in our case in order to get any nontrivial result.

Luckily, there is a well-known inequality of Grothendieck, which asserts that the value of the semidefinite program (1.2) and that of the integer program (1.1) can differ only by a constant factor. The precise value of this constant, called *Grothendieck's constant* and denoted by K_G , is not known, but it is known that its value is at most $\frac{\pi}{2 \ln(1+\sqrt{2})} = 1.782\dots$ (see [18]) and at least $\frac{\pi}{2} = 1.570\dots$ (see [11]). Stated in other words, the integrability gap of the problem is at most K_G . (Krivine mentions in [18] that he can improve the lower bound, but such an improvement has never been published.)

It follows that the value of the semidefinite program (1.2) provides an approximation of $\|A\|_{\infty \rightarrow 1}$ up to a constant factor. This, however, still does not tell us how to round the solution of the semidefinite program into an integral one with a comparable value. Indeed, this task requires more work and is carried out in the following sections.

We describe three rounding techniques. The first one is a deterministic procedure, which combines Grothendieck's inequality with some facts about four-wise independent random variables in a manner that resembles the technique used in [4] to approximate the second frequency moment of a stream of data under severe space constraints. The second rounding method is based on Rietz's proof of Grothendieck's inequality [24]. This proof supplies a better approximation guarantee for the special case of positive semidefinite matrices A , where the integrality gap can be shown to be precisely $\pi/2$, and implies that Nesterov's analysis for the problem he considers in [22] is tight.

The third technique, which supplies the best approximation guarantee, is based on Krivine's proof of Grothendieck's inequality. Here we use the vectors u_i, v_j which form a solution of the semidefinite program (1.2) to construct some other unit vectors u'_i, v'_j , which are first shown to exist in an infinite-dimensional Hilbert space and are then found, using another instance of semidefinite programming, in an $(n+m)$ -dimensional space. These vectors can now be rounded to $\{-1, 1\}$ in order to provide an integral solution for the original problem (1.1) in a rather simple way. We note that there are several known techniques for modifying the solution of a semidefinite program before rounding it; see [28], [19], [9]. Here, however, the modification seems more substantial.

We believe that the techniques presented here will have further applications, as they provide a method for handling problems in which there is a possible cancellation between positive and negative terms. It seems that there are additional interesting problems of this type. Approximation algorithms based on semidefinite programming for MAX CUT, MAX 2SAT, and related problems have been initiated in the seminal paper of Goemans and Williamson [12] and further developed in many subsequent papers. Unlike these problems, prior to the present paper there was no known polynomial time constant approximation algorithm for the problem considered here. Indeed, in this case, the semidefinite programming and its rounding appear to be essential in order to obtain any constant approximation guarantee, and not only in order to improve the constants ensured by appropriate combinatorial algorithms.

The rest of this paper is organized as follows. In section 2 we present the (relatively simple) proof that the problem of approximating the cut-norm $\|A\|_C$, as well as the related problem of approximating $\|A\|_{\infty \rightarrow 1}$, are both MAX SNP hard. In section 3 we describe a deterministic procedure that approximates the cut-norm of a given

matrix up to a constant factor. Two other methods, providing better constants, are described in section 4, where we also consider the special case of positive semidefinite matrices. Section 5 presents some examples which illustrate the relevance of the cut-norm to certain graph and matrix problems and shows how the results of this paper improve various known algorithms. We conclude with section 6, which includes some concluding remarks and open problems.

2. Hardness of approximation. As usual, we say that an approximation algorithm for a maximization problem has *performance ratio* or *performance guarantee* ρ for some real $\rho \leq 1$, if it always delivers a solution with objective function value at least ρ times the optimum value. Such an approximation algorithm is then called a ρ -*approximation algorithm*. Similarly, a randomized approximation algorithm is a ρ -approximation algorithm if it always produces a solution with expected value at least ρ times the optimum.

In this section we observe that the problem of approximating the cut-norm $\|A\|_C$ of a given input matrix is MAX SNP hard, and so is the related problem of approximating $\|A\|_{\infty \rightarrow 1}$. This implies, by the results in [23], [6], that there exists some $\rho < 1$ such that the existence of a ρ -approximation, polynomial time algorithm for any of these problems would imply that $P = NP$. The proof is by a reduction of the MAX CUT problem to the CUT NORM problem, and to the problem of computing $\|A\|_{\infty \rightarrow 1}$. We need the following simple observation.

LEMMA 2.1. *For any real matrix $A = (a_{ij})$,*

$$\|A\|_C \leq \|A\|_{\infty \rightarrow 1} \leq 4\|A\|_C.$$

Moreover, if the sum of each row and the sum of each column of A is zero, then $\|A\|_{\infty \rightarrow 1} = 4\|A\|_C$.

Proof. For any $x_i, y_j \in \{-1, 1\}$,

$$\sum_{i,j} a_{ij} x_i y_j = \sum_{\substack{i: x_i=1 \\ j: x_j=1}} a_{ij} - \sum_{\substack{i: x_i=1 \\ j: x_j=-1}} a_{ij} - \sum_{\substack{i: x_i=-1 \\ j: x_j=1}} a_{ij} + \sum_{\substack{i: x_i=-1 \\ j: x_j=-1}} a_{ij}.$$

The absolute value of each of the four terms in the right-hand side is at most $\|A\|_C$, implying, by the triangle inequality, that

$$(2.1) \quad \|A\|_{\infty \rightarrow 1} \leq 4\|A\|_C.$$

Suppose, say, that $\|A\|_C = \sum_{i \in I, j \in J} a_{ij}$ (the computation in the case where it equals $-\sum_{i \in I, j \in J} a_{ij}$ is essentially the same). Define $x_i = 1$ for $i \in I$ and $x_i = -1$ otherwise, and similarly, $y_j = 1$ if $j \in J$ and $y_j = -1$ otherwise. Then

$$\begin{aligned} \|A\|_C &= \sum_{i,j} a_{ij} \frac{1+x_i}{2} \frac{1+y_j}{2} \\ &= \frac{1}{4} \sum_{i,j} a_{ij} + \frac{1}{4} \sum_{i,j} a_{ij} x_i \cdot 1 + \frac{1}{4} \sum_{i,j} a_{ij} 1 \cdot y_j + \frac{1}{4} \sum_{i,j} a_{ij} x_i y_j. \end{aligned}$$

The absolute value of each of the four terms in the right-hand side is at most $\|A\|_{\infty \rightarrow 1}/4$, implying that $\|A\|_{\infty \rightarrow 1} \geq \|A\|_C$. If the sum of each row and the sum of each column of A is zero, then the right-hand side is precisely $\frac{1}{4} \sum_{i,j} a_{ij} x_i y_j$, implying that in this case $\|A\|_{\infty \rightarrow 1} \geq 4\|A\|_C$, which, in view of (2.1), shows that the above holds as an equality. \square

PROPOSITION 2.2. *Given a (weighted or unweighted) graph $G = (V, E)$, there is an efficient way to construct a real $2|E|$ by $|V|$ matrix A , such that*

$$\text{MAXCUT}(G) = \|A\|_C = \|A\|_{\infty \mapsto 1} / 4.$$

Therefore, the CUT NORM problem and the problem of computing $\|A\|_{\infty \mapsto 1}$ are both MAX SNP hard.

Proof. We describe the construction for the unweighted case. The weighted case is similar. Given $G = (V, E)$, orient it in an arbitrary manner. Let $V = \{v_1, v_2, \dots, v_n\}$ and $E = \{e_1, e_2, \dots, e_m\}$, and let $A = (a_{ij})$ be the $2m$ by n matrix defined as follows. For each $1 \leq i \leq m$, if e_i is oriented from v_j to v_k , then $a_{2i-1,j} = a_{2i,k} = 1$ and $a_{2i-1,k} = a_{2i,j} = -1$. The rest of the entries of A are all 0. It is not difficult to check that $\text{MAXCUT}(G) = \|A\|_C$. In addition, since the sum of entries in each row and in each column of A is zero, it follows by Lemma 2.1 that $\|A\|_{\infty \mapsto 1} = 4\|A\|_C$. As it is known [23], [14] that the MAX CUT problem is MAX SNP hard, the desired result follows. \square

Håstad [14] has shown that if $P \neq NP$, then there is no polynomial-time approximation algorithm for the MAX CUT problem with approximation ratio exceeding $16/17$. Thus, this is an upper bound for the best possible approximation guarantee of a polynomial algorithm for approximating $\|A\|_C$ or $\|A\|_{\infty \mapsto 1}$. Similarly, our reduction above can be easily modified to construct, for any given directed graph D , a matrix B with vanishing row sums and column sums, so that the value of the maximum directed cut of D is equal to $\|B\|_C$. Håstad [14] has shown that if $P \neq NP$, then there is no polynomial-time approximation algorithm for the MAX DICUT problem with approximation ratio exceeding $12/13$. Thus, this is an upper bound for the best possible approximation guarantee of a polynomial approximation algorithm for $\|B\|_C$ or $\|B\|_{\infty \mapsto 1}$.

3. Approximating the cut-norm. In this section we describe an efficient, deterministic, ρ -approximation algorithm for the CUT NORM problem, where $\rho > 0$ is an absolute constant. We make no attempt here to optimize the value of ρ : this will be done (in a different way) in section 4. We believe, however, that although the value of ρ obtained in this section is rather poor, the method, which is motivated by the proof of Grothendieck’s inequality in [7, p. 15] and [16, p. 68], is interesting and may lead to similar results for related problems.

Given a real n by m matrix $A = (a_{ij})$, our objective is to find $x_i, y_j \in \{-1, 1\}$, such that

$$\sum_{i,j} a_{ij} x_i y_j \geq \rho \|A\|_{\infty \mapsto 1},$$

where ρ is an absolute positive constant. The discussion in section 1 and the proof of Lemma 2.1 imply that this will yield a similar procedure for finding I and J such that $|\sum_{i \in I, j \in J} a_{ij}| \geq \rho' \|A\|_C$.

We start by solving the semidefinite program (1.2). We can thus compute, for any positive δ , unit vectors $u_i, v_j \in R^p$, where $p = n + m$, such that the sum $\sum_{i,j} a_{ij} u_i \cdot v_j$ is at least the maximum value of the program (1.2) (which is clearly at least $\|A\|_{\infty \mapsto 1}$) minus δ . Since the value of the above norm of A is at least the maximum absolute value of an entry of A , we can make sure that the δ term is negligible. The main part of the algorithm is the rounding phase, that is, the phase of finding, using the vectors

u_i, v_j , reals $x_i, y_j \in \{-1, 1\}$, such that

$$\sum_{i,j} a_{ij} x_i y_j \geq \rho \sum_{i,j} a_{ij} u_i \cdot v_j.$$

This is done as follows. Let V be an explicit set of $t = O(p^2)$ vectors $\epsilon = (\epsilon_1, \epsilon_2, \dots, \epsilon_p) \in \{-1, 1\}^p$ in which the values ϵ_j are four-wise independent and each of them attains the two values -1 and 1 with equal probability. This means that for every four distinct coordinates $1 \leq i_1 < \dots < i_4 \leq n$ and every choice of $\epsilon_1, \dots, \epsilon_4 \in \{-1, 1\}$, exactly a $(1/16)$ -fraction of the vectors have ϵ_j in their coordinate number i_j for $j = 1, \dots, 4$. As described, for example, in [1] or [5] such sets (also known as *orthogonal arrays of strength 4*) can be constructed efficiently using the parity check matrices of BCH codes.

Let $M > 0$ be a fixed real, to be chosen later. Consider V as a sample space in which all t points ϵ have the same probability. For any unit vector $q = (q_1, q_2, \dots, q_p) \in R^p$, let $H(q)$ denote the random variable defined on the sample space V by putting $[H(q)](\epsilon) = \sum_{j=1}^p \epsilon_j q_j$. Since the entries ϵ_j are four-wise (and hence pairwise) independent, it follows that for any two vectors q, q' , the expectation of $H(q)H(q')$ is precisely the inner product $q \cdot q'$. In particular, the expectation of $[H(q)]^2$ is $q \cdot q = 1$. Similarly, four-wise independence implies that the expectation of $[H(q)]^4$ satisfies

$$E([H(q)]^4) = \sum_{j=1}^p q_j^4 + 6 \sum_{1 \leq j < j' \leq p} q_j^2 q_{j'}^2 \leq 3 \left(\sum_j q_j^2 \right)^2 = 3.$$

The M -truncation of $H(q)$, denoted $H^M(q)$, is defined as follows: $[H^M(q)](\epsilon) = [H(q)](\epsilon)$ if $|[H(q)](\epsilon)| \leq M$, $[H^M(q)](\epsilon) = M$ if $[H(q)](\epsilon) > M$, and $[H^M(q)](\epsilon) = -M$ if $[H(q)](\epsilon) < -M$.

By Markov's inequality, for every positive real m ,

$$\text{Prob}[|H(q)| \geq m] \cdot m^4 \leq E([H(q)]^4) \leq 3,$$

implying that $\text{Prob}[|H(q)| \geq m] \leq \frac{3}{m^4}$. This implies the following.

CLAIM 3.1. *The expectation $E(|H(q) - H^M(q)|^2)$ satisfies*

$$E(|H(q) - H^M(q)|^2) \leq \frac{1}{M^2}.$$

Proof. For every nonnegative random variable X we have that $EX^2 = 2 \int_0^\infty u \cdot \text{Prob}(X \geq u) du$. Hence,

$$\begin{aligned} E(|H(q) - H^M(q)|^2) &= 2 \int_0^\infty u \cdot \text{Prob}(|H(q)| \geq M + u) du \\ &\leq \int_0^\infty \frac{6u}{(M + u)^4} du = \frac{1}{M^2}. \quad \square \end{aligned}$$

Each random variable $H(q)$ can be associated with a vector $h(q) \in R^t$ by defining $[h(q)](\epsilon) = \frac{1}{\sqrt{t}} [H(q)](\epsilon)$. The truncation $h^M(q) = H^M(q)/\sqrt{t}$ is defined analogously. The above discussion thus implies the following.

LEMMA 3.2. *For each unit vector $q \in R^p$, $h(q) \in R^t$ is a unit vector. The norm of $h^M(q)$ is at most 1, and that of $h(q) - h^M(q)$ is at most $1/M$. If $q' \in R^p$ is another vector, then $h(q) \cdot h(q') = q \cdot q'$.*

Returning to our semidefinite program (1.2) and its solution (up to δ) given by the vectors $u_i, v_j \in R^p$, let B denote the value of the program. Since $h(q) \cdot h(q') = q \cdot q'$ for all unit vectors q, q' , it follows that

$$\begin{aligned} B - \delta &\leq \sum_{ij} a_{ij} u_i \cdot v_j \\ &= \sum_{ij} a_{ij} h(u_i) \cdot h(v_j) \\ &= \sum_{ij} a_{ij} h^M(u_i) \cdot h^M(v_j) + \sum_{ij} a_{ij} (h(u_i) - h^M(u_i)) \cdot h^M(v_j) \\ &\quad + \sum_{ij} a_{ij} h(u_i) \cdot (h(v_j) - h^M(v_j)). \end{aligned}$$

By the convexity of the program, and since the norm of each vector $h^M(v_j), h(u_i)$ is at most 1 and the norm of each vector $h(u_i) - h^M(u_i)$ and each vector $h(v_j) - h^M(v_j)$ is at most $1/M$, it follows that

$$\sum_{ij} a_{ij} (h(u_i) - h^M(u_i)) \cdot h^M(v_j) + \sum_{ij} a_{ij} h(u_i) \cdot (h(v_j) - h^M(v_j)) \leq \frac{2}{M} B.$$

Here we have used, crucially, the fact that as B is the maximum value of the semidefinite program, its value on the vectors $h(u_i) - h^M(u_i)$ and the vectors $h(v_j)$ does not exceed $B \frac{1}{M}$, and so does its value on the vectors $h(u_i)$ and $h(v_j) - h^M(v_j)$. Therefore,

$$B \left(1 - \frac{2}{M}\right) - \delta \leq \sum_{ij} a_{ij} h^M(u_i) \cdot h^M(v_j).$$

It follows that there is a coordinate $\epsilon \in V$ such that

$$\sum_{ij} a_{ij} h^M(u_i)(\epsilon) \cdot h^M(v_j)(\epsilon) \geq \frac{1}{t} \left(B \left(1 - \frac{2}{M}\right) - \delta \right).$$

By the definition of the vectors h , this implies

$$\sum_{ij} a_{ij} H^M(u_i)(\epsilon) \cdot H^M(v_j)(\epsilon) \geq B \left(1 - \frac{2}{M}\right) - \delta.$$

Choose $M = 3$ and define $x_i = \frac{H^M(u_i)(\epsilon)}{M}, y_j = \frac{H^M(v_j)(\epsilon)}{M}$. Then x_i, y_j are reals, each having an absolute value at most 1, and

$$\sum_{ij} a_{ij} x_i y_j \geq B \left(\frac{M-2}{M^3}\right) - \frac{\delta}{M^2} = \frac{B}{27} - \frac{\delta}{9}.$$

Fixing all x_i, y_j but, say, x_1 , the left-hand side is a linear form in x_1 , and thus we can shift x_1 to either -1 or 1 , without any decrease in the value of the sum. Proceeding in this way with the other variables, each one in its turn, we obtain $x_i, y_j \in \{-1, 1\}$ such that the value of the sum $\sum_{ij} a_{ij} x_i y_j$ is at least $\frac{B}{27} - \frac{\delta}{9}$. As δ is arbitrarily small, we have thus proved the following.

THEOREM 3.3. *There is a deterministic polynomial-time algorithm that finds, for a given real matrix $A = (a_{ij})$, integers $x_i, y_j \in \{-1, 1\}$ such that the value of the sum $\sum_{ij} a_{ij} x_i y_j$ is at least $0.03 B$, where B is the value of the semidefinite program (1.2) (which is at least $\|A\|_{\infty \rightarrow 1}$).*

4. Improving the constant. The constant obtained in the previous section can be improved by replacing the space V of four-wise independent $\{-1, 1\}$ variables with a space of $2k$ -wise independent $\{-1, 1\}$ variables (or with a space of independent standard normal random variables). This, however, will not provide a ρ -approximation algorithm with $\rho > \frac{1}{5}$.

In fact, we can do much better. In this section we describe two randomized ρ -approximation algorithms for approximating $\|A\|_{\infty \rightarrow 1}$. For the first algorithm, $\rho = \frac{4}{\pi} - 1 > 0.27$, while for the second $\rho = \frac{2 \ln(1+\sqrt{2})}{\pi} > 0.56$. We further show that these yield randomized ρ -approximation algorithms for the CUT NORM problem, with the same values of ρ (without losing the factor of 4 that appears in Lemma 2.1). It should be possible to derandomize these algorithms using the techniques of [21].

4.1. Averaging with a Gaussian measure: Rietz’s method. The main idea here, based on [24], is to round the solution $u_i, v_j \in R^p$ of the semidefinite program (1.2) by averaging over R^p with normalized Gaussian measure. We proceed with the details.

Let g_1, g_2, \dots, g_p be standard, independent, Gaussian random variables, and consider the random Gaussian vector $G = (g_1, \dots, g_p)$. The following identity holds for every two unit vectors $b, c \in \ell_2^p$:

$$(4.1) \quad \frac{\pi}{2} E [\text{sign}(b \cdot G) \cdot \text{sign}(c \cdot G)] \\ = b \cdot c + E \left\{ \left[b \cdot G - \sqrt{\frac{\pi}{2}} \text{sign}(b \cdot G) \right] \cdot \left[c \cdot G - \sqrt{\frac{\pi}{2}} \text{sign}(c \cdot G) \right] \right\}.$$

This is a simple exercise, using rotation invariance. Indeed, the fact that

$$(4.2) \quad E [(b \cdot G)(c \cdot G)] = b \cdot c$$

follows from the fact that the random variables g_i are uncorrelated; if $b = (b_1, \dots, b_p)$ and $c = (c_1, \dots, c_p)$, then

$$E [(b \cdot G)(c \cdot G)] = E \left[\sum_{i=1}^p b_i g_i \sum_{i=1}^p c_i g_i \right] = \sum_{i,j} b_i c_j E [g_i g_j] = \sum_{i=1}^p b_i c_i = b \cdot c.$$

To compute $E [(b \cdot G)\text{sign}(c \cdot G)]$ we may assume, by rotation invariance, that $c = (1, 0, \dots, 0)$ and $b = (b_1, b_2, 0, \dots, 0)$. Hence $b \cdot c = b_1$ and

$$\begin{aligned} E [(b \cdot G)\text{sign}(c \cdot G)] &= E [(b_1 g_1 + b_2 g_2)\text{sign}(g_1)] \\ &= E [b_1 g_1 \text{sign}(g_1)] + E [b_2 g_2] E [\text{sign}(g_1)] \\ &= E [b_1 g_1 \text{sign}(g_1)] \\ &= 2 \int_0^\infty \frac{1}{\sqrt{2\pi}} b_1 x e^{-x^2/2} dx = \sqrt{\frac{2}{\pi}} b_1. \end{aligned}$$

Thus

$$(4.3) \quad E [(b \cdot G)\text{sign}(c \cdot G)] = \sqrt{\frac{2}{\pi}} b \cdot c.$$

The identity above follows from (4.2) and (4.3) by linearity of expectation.

Suppose now that the vectors $u_i, v_j \in R^p$ supply a solution of (1.2), which can be found efficiently. (We assume, for simplicity, that this is a precise solution.) Let B denote the value of the solution. By (4.1),

$$(4.4) \quad \frac{\pi}{2} E \left\{ \sum_{i,j} a_{ij} [\text{sign}(u_i \cdot G) \cdot \text{sign}(v_j \cdot G)] \right\} \\ = B + \sum_{i,j} a_{ij} E \left\{ \left[u_i \cdot G - \sqrt{\frac{\pi}{2}} \cdot \text{sign}(u_i \cdot G) \right] \cdot \left[v_j \cdot G - \sqrt{\frac{\pi}{2}} \cdot \text{sign}(v_j \cdot G) \right] \right\}.$$

Note that each term of the form

$$E \left\{ \left[u_i \cdot G - \sqrt{\frac{\pi}{2}} \cdot \text{sign}(u_i \cdot G) \right] \cdot \left[v_j \cdot G - \sqrt{\frac{\pi}{2}} \cdot \text{sign}(v_j \cdot G) \right] \right\},$$

which multiplies a_{ij} in (4.4), is the inner product of two vectors in a Hilbert space. Moreover, the square of the norm of each of these vectors is easily seen to be $\frac{\pi}{2} - 1$ by substituting $b = c = u_i$ (or $b = c = v_j$) in (4.1). Therefore, as B is the maximum possible value of the program (1.2), it follows that the last sum in (4.4) is bounded, in absolute value, by $(\frac{\pi}{2} - 1)B$. Thus, by (4.4),

$$\frac{\pi}{2} E \left\{ \sum_{i,j} a_{ij} [\text{sign}(u_i \cdot G) \cdot \text{sign}(v_j \cdot G)] \right\} \geq \left(2 - \frac{\pi}{2} \right) B,$$

implying that by choosing the normal random variables g_i randomly and by defining

$$x_i = \text{sign}(u_i \cdot G), \quad y_j = \text{sign}(v_j \cdot G)$$

we get a solution for (1.1) whose expected value is at least $\frac{2}{\pi}(2 - \frac{\pi}{2})B = (\frac{4}{\pi} - 1)B$. As B is at least the value of the optimal solution of (1.1), this supplies a randomized ρ -approximation algorithm for estimating $\|A\|_{\infty \rightarrow 1}$, where $\rho = \frac{4}{\pi} - 1$.

Remark. The above algorithm is based only on the basic idea in the proof of [24], and it is in fact possible to improve its performance guarantee by modifying it according to the full proof. This suggests rounding $u_i \cdot G$ to $x_i = \text{sign}(u_i \cdot G)$ if it is “large” and rounding it to some multiple of $u_i \cdot G$ if it is not, where the definition of “large” and the precise multiple are chosen optimally. We can then further round the fractional values of x_i to integral ones with no loss. This resembles some of the ideas used in several recent papers, including [28], [19], and [9]. We do not include the details here, as we can get a better approximation guarantee by using another method described in subsection 4.3.

4.2. Positive semidefinite matrices. In this section we observe that when $A = (a_{ij})$ is positive semidefinite, then the approximation ratio can be improved to $2/\pi$. This follows from the work of Nesterov [22], but the short argument we describe here is based on Riesz’s proof that for such an A the constant in Grothendieck’s inequality can be improved to $\pi/2$. Grothendieck himself showed in [11] (in a somewhat different language) that $\pi/2$ is a lower bound for the constant in this case; we sketch a proof of this fact below.

We first show that if $A = (a_{ij})$ is a positive semidefinite n by n matrix, then the maximum of the semidefinite program (1.2) is obtained for some vectors u_i, v_j

satisfying $u_i = v_i$ for all i (though, of course, it may also be obtained for some vectors that do not satisfy this property). This is not really required for getting the $2/\pi$ -approximation algorithm, but we include this proof as it shows that the integrality gap of the problem is at most $\pi/2$.

Suppose thus that the maximum value of (1.2), denoted by B , is obtained by some p -dimensional vectors, where $p \leq 2n$ and A is positive semidefinite. Let $D = A \otimes I$ be the tensor product of A with a p by p identity matrix. This is simply the np by np matrix consisting of p blocks along the diagonal, each being a copy of A , so that, in particular, D is positive semidefinite. Given $u_1, \dots, u_n \in \ell_2^p$, where $u_i = (u_{i1}, u_{i2}, \dots, u_{ip})$, let $u^{(j)} = (u_{1j}, u_{2j}, \dots, u_{nj})$ be the vector consisting of the j th coordinates of all vectors u_i ($1 \leq j \leq p$). Let $u = (u^{(1)}, u^{(2)}, \dots, u^{(p)}) \in \ell_2^{np}$ and note that $Du = (Au^{(1)}, Au^{(2)}, \dots, Au^{(p)})$. Thus D is positive semidefinite and for $u_1, \dots, u_n, v_1, \dots, v_n \in \ell_2^p$, with v defined analogously to u ,

$$\sum_{i,j} a_{ij} u_i \cdot v_j = Du \cdot v = D^{1/2}u \cdot D^{1/2}v \leq \|D^{1/2}u\| \cdot \|D^{1/2}v\|,$$

with equality when $u = v$.

Therefore, if the maximum B of the quantity $\sum_{i,j} a_{ij} u_i \cdot v_j$ is obtained for the unit vectors $u_i, v_j \in \ell_2^p$, then, as $\|D^{1/2}u\|$ cannot exceed $B^{1/2}$, it follows that $\|D^{1/2}u\| = \|D^{1/2}v\| = B^{1/2}$. Thus the maximum is also equal to $\sum_{i,j} a_{ij} u_i \cdot u_j$ (and to $\sum_{i,j} a_{ij} v_i \cdot v_j$).

The fact that for positive semidefinite matrices A we can get a ρ -approximation algorithm with $\rho = \frac{2}{\pi}$ is now an easy consequence of (4.4). We first solve the semidefinite program (1.2) to get vectors u_i, v_j optimizing it. By the above discussion, the vectors $u_i = v_i$ for all i also give an optimal solution. (Alternatively, we can solve the variant of (1.2) in which $u_i = v_i$ for all i directly, and proceed from there.) By (4.4),

$$\begin{aligned} & \frac{\pi}{2} E \left\{ \sum_{i,j} a_{ij} [\text{sign}(u_i \cdot G) \cdot \text{sign}(u_j \cdot G)] \right\} \\ &= B + \sum_{i,j} a_{ij} E \left\{ \left[u_i \cdot G - \sqrt{\frac{\pi}{2}} \cdot \text{sign}(u_i \cdot G) \right] \cdot \left[u_j \cdot G - \sqrt{\frac{\pi}{2}} \cdot \text{sign}(u_j \cdot G) \right] \right\} \geq B, \end{aligned}$$

where here we have used the fact that A is positive semidefinite. The algorithm now simply chooses G at random, and computes $x_i = y_i = \text{sign}(u_i \cdot G)$.

We conclude this subsection with a sketch of (a modified version of) the argument of Grothendieck that shows that the integrality gap of (1.1) for the positive semidefinite case is indeed precisely $\pi/2$. We need the following.

Fact. If b and c are two random, independent vectors on the unit sphere in R^p , then

$$(4.5) \quad E[(b \cdot c)^2] = \frac{1}{p}$$

and

$$(4.6) \quad E[|b \cdot c|] = \left(\sqrt{\frac{2}{\pi}} + o(1) \right) \frac{1}{\sqrt{p}},$$

where the $o(1)$ term tends to zero as p tends to infinity.

Indeed, the computation of the first expectation is very simple; by rotation invariance we may assume that $c = (1, 0, 0, \dots, 0)$ and then $E[(b \cdot c)^2] = E[b_1^2]$, where we write $b = (b_1, b_2, \dots, b_p)$. However, by symmetry $E[b_1^2] = \frac{1}{p}E[b_1^2 + b_2^2 + \dots + b_p^2] = \frac{1}{p}$, implying (4.5). By the same reasoning, $E[|b \cdot c|] = E[|b_1|]$ which can now either be computed directly by integrating along the sphere or be estimated by noticing that it is well approximated by $E[|\frac{g}{\sqrt{p}}|]$, where g is a standard Gaussian random variable. The simple computation for this case appears before the derivation of (4.3).

Armed with the above fact, we now define an n by n positive semidefinite matrix $A = (a_{ij})$ for which the ratio between the value of (1.1) and that of (1.2) is (nearly) $\pi/2$. Fix a large integer p , and let n be much larger. Let v_1, v_2, \dots, v_n be n independent random vectors chosen uniformly in the unit sphere in R^p . Let A be the Gram matrix of the vectors $\frac{v_i}{n}$, that is, $a_{ij} = \frac{1}{n^2}v_i \cdot v_j$. Obviously A is positive semidefinite. Moreover, if we substitute the unit vectors v_i in the program (1.2) we get

$$\sum_{i,j} a_{ij}v_i \cdot v_j = \frac{1}{n^2} \sum_{ij} (v_i \cdot v_j)^2.$$

When n tends to infinity, this converges to the average value of the square of the inner product between two random vectors on the unit sphere of R^p , which is, by (4.5), $1/p$. Therefore, the optimal value of the program (1.2) for A is at least $1/p$.

Consider, now, the optimal value of the integer program (1.1) for A . Let $x_i \in \{-1, 1\}$ be a sign vector. Then

$$\sum_{i,j} a_{ij}x_i \cdot x_j = \left\| \frac{1}{n} \sum_{i=1}^n x_i v_i \right\|^2.$$

Therefore, the value of the integer program is the square of the maximum possible norm of a vector $\frac{1}{n} \sum_{i=1}^n x_i v_i$, where $x_i \in \{-1, 1\}$ for all i . If the direction of this optimal vector is given by the unit vector c , then, knowing c , it is clear how to choose x_i for each i ; it simply has to be $\text{sign}(v_i \cdot c)$. With this choice of the signs x_i , the quantity

$$\frac{1}{n} \sum_{i=1}^n x_i v_i \cdot c = \left\| \frac{1}{n} \sum_{i=1}^n x_i v_i \right\|$$

converges, when n tends to infinity, to the average value of $|v \cdot c|$, where v is a random vector on the sphere. By (4.6) this value is $(\sqrt{2/\pi} + o(1))\frac{1}{\sqrt{p}}$, where the $o(1)$ term tends to zero as p tends to infinity. Since n can be chosen to be arbitrarily large with respect to p , and as we do not have to consider all the infinitely many possible directions c but can consider an appropriate ϵ -net of directions on the sphere, we conclude that if p is large and n is huge, then, with high probability, the value of the integer program (1.1) for A is at most

$$\left[\left(\sqrt{\frac{2}{\pi}} + o(1) \right) \frac{1}{\sqrt{p}} \right]^2 = \left(\frac{2}{\pi} + o(1) \right) \frac{1}{p}.$$

It follows that the integrality gap is at least $\pi/2 - o(1)$, implying, by the discussion in the beginning of this subsection, that it is $\pi/2 + o(1)$.

4.3. Constructing new vectors: Krivine’s argument. The approach in this subsection is based on a simplified version of Krivine’s proof, presented in [18], of Grothendieck’s inequality as described in [15]. We need the following simple lemma, which has already been mentioned briefly in the introduction, and which has been applied in Grothendieck’s original proof as well.

LEMMA 4.1 (Grothendieck’s identity). *For every two unit vectors u, v in a finite-dimensional Euclidean space, if z is chosen randomly and uniformly from the unit sphere of the space, then*

$$\frac{\pi}{2} \cdot E([\text{sign}(u \cdot z)] \cdot [\text{sign}(v \cdot z)]) = \arcsin(u \cdot v).$$

Using this lemma, we prove the following.

LEMMA 4.2. *For any set $\{u_i : 1 \leq i \leq n\} \cup \{v_j : 1 \leq j \leq m\}$ of unit vectors in a Hilbert space H , and for $c = \sinh^{-1}(1) = \ln(1 + \sqrt{2})$, there is a set $\{u'_i : 1 \leq i \leq n\} \cup \{v'_j : 1 \leq j \leq m\}$ of unit vectors in a finite-dimensional Hilbert space H' , such that if z is chosen randomly and uniformly in the unit sphere of H' , then*

$$\frac{\pi}{2} \cdot E([\text{sign}(u'_i \cdot z)] \cdot [\text{sign}(v'_j \cdot z)]) = c u_i \cdot v_j$$

for all $1 \leq i \leq n, 1 \leq j \leq m$.

Proof. Fix c as above, a Hilbert space H , and $u, v \in H$. By Taylor’s expansion,

$$\sin(c u \cdot v) = \sum_{k=0}^{\infty} (-1)^k \frac{c^{2k+1}}{(2k+1)!} (u \cdot v)^{2k+1}.$$

For every vector w and integer j denote by $w^{\otimes j}$ the j th tensor power $w \otimes w \otimes \dots \otimes w$ (j terms). Then the above expansion becomes

$$\sin(c u \cdot v) = \sum_{k=0}^{\infty} (-1)^k \frac{c^{2k+1}}{(2k+1)!} u^{\otimes(2k+1)} \cdot v^{\otimes(2k+1)}.$$

Consider the following vectors in the direct sum $\bigoplus_{k=0}^{\infty} H^{\otimes(2k+1)}$, whose k th “coordinates” are given by

$$T(u)_k = (-1)^k \sqrt{\frac{c^{2k+1}}{(2k+1)!}} \cdot u^{\otimes(2k+1)} \quad \text{and} \quad S(v)_k = \sqrt{\frac{c^{2k+1}}{(2k+1)!}} \cdot v^{\otimes(2k+1)}.$$

Then the above expansion boils down to $\sin(c u \cdot v) = T(u) \cdot S(v)$, or

$$c u \cdot v = \arcsin(T(u) \cdot S(v)).$$

Moreover,

$$\|T(u)\|^2 = \sinh(c \cdot \|u\|^2) \quad \text{and} \quad \|S(v)\|^2 = \sinh(c \cdot \|v\|^2).$$

Given the unit vectors u_i, v_j , recall that $c = \sinh^{-1}(1)$ and define $u'_i = T(u_i)$ and $v'_j = S(v_j)$. Note that all the vectors u'_i, v'_j are unit vectors (in the huge direct sum of tensor products we constructed). Let H' be the span of u'_i, v'_j . It is an $(m+n)$ -dimensional Hilbert space. Let z be a random vector chosen uniformly on its unit sphere. By Grothendieck’s identity (Lemma 4.1), for every i, j ,

$$\frac{\pi}{2} \cdot E([\text{sign}(T(u_i) \cdot z)] \cdot [\text{sign}(S(v_j) \cdot z)]) = \arcsin(T(u_i) \cdot S(v_j)) = c u_i \cdot v_j$$

as needed. \square

THEOREM 4.3. *There is a randomized polynomial-time algorithm that, given an input n by m matrix $A = (a_{ij})$ and unit vectors u_i, v_j in R^{n+m} , finds $x_i, y_j \in \{-1, 1\}$ such that the expected value of the sum $\sum_{ij} a_{ij} x_i y_j$ is*

$$\frac{2 \ln(1 + \sqrt{2})}{\pi} \sum_{ij} a_{ij} u_i \cdot v_j.$$

Therefore, there is a polynomial randomized ρ -approximation algorithm for computing $\|A\|_{\infty \mapsto 1}$, where $\rho = \frac{2 \ln(1 + \sqrt{2})}{\pi} > 0.56$.

Proof. By Lemma 4.2 there are vectors u'_i, v'_j satisfying the conclusion of the lemma. We can thus find such vectors in an $(m+n)$ -dimensional space, using semidefinite programming. (This step can in fact be performed in a more efficient way either by using enough coordinates of the infinite-dimensional vectors $T(u_i), S(v_j)$ defined in the proof of Lemma 4.2 or by finding a root of the Gram matrix of the vectors u'_i, v'_j whose terms can be computed by the same proof. This, however, will not change the total running time by more than a constant factor, as we still have to solve one semidefinite programming problem for finding the vectors u_i, v_j .) By linearity of expectation, with $c = \sinh^{-1}(1)$ as above,

$$c \cdot \sum_{i,j} a_{ij} u_i \cdot v_j = \frac{\pi}{2} \cdot E \left(\sum_{i,j} a_{ij} [\text{sign}(u'_i \cdot z)] \cdot [\text{sign}(v'_j \cdot z)] \right).$$

We can now simply pick a random z and define $x_i = \text{sign}(u'_i \cdot z)$ and $y_j = \text{sign}(v'_j \cdot z)$. \square

4.4. The cut-norm. In this short subsection we observe that the same approximation ratio guaranteed in any approximation algorithm for $\|A\|_{\infty \mapsto 1}$ can be obtained for the CUT NORM problem as well. Given an n by m matrix $A = a_{ij}$, augment it to an $(n+1)$ by $(m+1)$ matrix $A' = (a'_{ij})$ by defining $a'_{ij} = a_{ij}$ for all $1 \leq i \leq n, 1 \leq j \leq m, a'_{i,m+1} = -\sum_{j=1}^m a_{ij}$ for all $1 \leq i \leq n, a'_{n+1,j} = -\sum_{i=1}^n a_{ij}$ for all $1 \leq j \leq m$, and $a'_{n+1,m+1} = 0$. We claim that $\|A'\|_C = \|A\|_C$. Indeed, obviously $\|A'\|_C \geq \|A\|_C$, as A is a submatrix of A' . Conversely, let $I \subset \{1, 2, \dots, n+1\}, J \subset \{1, 2, \dots, m+1\}$ satisfy $\|A'\|_C = |\sum_{i \in I, j \in J} a'_{ij}|$. If $n+1 \in I$, replace it by its complement $\{1, 2, \dots, n+1\} \setminus I$, and similarly, if $m+1 \in J$, replace it by its complement $\{1, 2, \dots, m+1\} \setminus J$. As the sum of each row and the sum of each column of A' is zero, the absolute value of the sum $\sum_{i \in I, j \in J} a'_{ij}$ with the new sets I, J is still equal to $\|A'\|_C$. This is, however, the sum of elements of a submatrix of A , implying that in fact $\|A'\|_C = \|A\|_C$ as claimed.

By the last part of Lemma 2.1, $\|A'\|_{\infty \mapsto 1} = 4\|A'\|_C$, and thus we can simply apply any algorithm for ρ -approximating $\|A'\|_{\infty \mapsto 1}$ to obtain a similar ρ -approximation of $\|A'\|_C = \|A\|_C$.

5. Examples and motivation. In order to explain the motivation for finding approximation algorithms for the cut-norm, in this section we present a few illustrative examples showing how the cut-norm occurs naturally in algorithmic contexts.

Let $G = (V, E)$ be an undirected graph, and let A and B be two disjoint nonempty subsets of V . Let $e(A, B)$ denote the number of edges of G with an endpoint in A and an endpoint in B , and define the *density of edges* between A and B by $d(A, B) = \frac{e(A, B)}{|A||B|}$. For $\epsilon > 0$, the pair (A, B) is called ϵ -regular if for every $X \subset A$ and $Y \subset B$

satisfying $|X| \geq \epsilon|A|$ and $|Y| \geq \epsilon|B|$, we have

$$|d(A, B) - d(X, Y)| < \epsilon.$$

The regularity lemma of Szemerédi [25] is a fundamental result that asserts that any graph can be partitioned in a certain regular way. An algorithmic version of this lemma appears in [3], together with several algorithmic applications. The main step in [3] is a polynomial-time algorithm that, given two disjoint subsets $A, B \subset V$ in a graph G , where $|A| = |B| = n$ and given $\epsilon > 0$, either decides that the pair (A, B) is ϵ -regular or finds two subsets $X \subset A$ and $Y \subset B$, each of size at least $\frac{\epsilon}{16}n$, such that $|d(A, B) - d(X, Y)| \geq \epsilon^4$. Given $G = (V, E)$, $\epsilon > 0$, and A, B as above, denote $d = d(A, B)$. Let $F = (f_{ab})_{a \in A, b \in B}$ be the n by n matrix defined by $f_{ab} = 1 - d$ if $ab \in E$ and $f_{ab} = -d$ if $ab \notin E$. Note that if (A, B) is not ϵ -regular, then there are $I \subset A$, $J \subset B$ satisfying $|I| \geq \epsilon n$, $|J| \geq \epsilon n$ such that

$$\left| \sum_{a \in I, b \in J} f_{ab} \right| \geq \epsilon|I||J| \geq \epsilon^3 n^2,$$

that is, the cut-norm of F is at least $\epsilon^3 n^2$. Therefore, in this case the algorithm presented in this paper will find efficiently $X \subset A$, $Y \subset B$ such that

$$\left| \sum_{a \in X, b \in Y} f_{ab} \right| \geq 0.56\epsilon^3 n^2.$$

Obviously this implies, say, that $|X| \geq 0.5\epsilon^3 n$, $|Y| \geq 0.5\epsilon^3 n$, and $|d(X, Y) - d(A, B)| \geq 0.5\epsilon^3$.

If the algorithm does not find such sets, it can report that the pair (A, B) is ϵ -regular. This can be used instead of Corollary 3.3 in [3] to obtain efficiently a regular partition of any given graph with less parts than the ones ensured by the algorithm in [3]. (But the number will still be huge; a tower of height polynomial in $1/\epsilon$, the degree of this polynomial will be smaller than the one in [3]. By being a bit careful this height can be shown to be $O(1/\epsilon^7)$, whereas the height obtained in [3] is $\Theta(1/\epsilon^{20})$.) Moreover, our algorithm actually finds subsets $I \subset A$, $J \subset B$ maximizing (approximately) the value of $|e(I, J)| \cdot |A| \cdot |B| - e(A, B)|I| \cdot |J|$, while the algorithm in [3] relies on the fact that this maximum is of order n^2 (i.e., that the problem is “dense”).

The rounding techniques described in section 3 or in subsection 4.1 (but not the one described in subsection 4.3) can be used to find efficiently, for any given square n by n matrix $A = (a_{ij})$, a vector $(x_1, \dots, x_n) \in \{-1, 1\}$, such that the value of $|\sum_{ij} a_{ij} x_i x_j|$ is at least a ρ -fraction of the maximum possible value of this quantity (or even the quantity $|\sum_{ij} a_{ij} u_i \cdot u_j|$, where u_i, u_j are unit vectors in a Hilbert space). Note that here we do not assume that A is positive semidefinite (but we try to maximize the absolute value of the quadratic form, not the quadratic form itself). By applying this approximation algorithm to a matrix defined from a graph G as above, we can find an induced subgraph that approximates the maximum possible deviation of the total number of edges from its expected value among all induced subgraphs of the graph.

In [8] Frieze and Kannan describe an efficient algorithm for finding what they call a *cut-decomposition* of a given n by m real matrix A , and apply it to obtain efficient approximation algorithms for various dense graph and matrix problems.

Consider matrices with a set of rows indexed by R and a set of columns indexed by S . For $I \subset R$ and $J \subset S$, and for a real d , the *cut matrix* $D = CUT(I, J, d)$ is the matrix $(d_{ij})_{i \in R, j \in S}$ defined by $d_{ij} = d$ if $i \in I, j \in J$ and $d_{ij} = 0$ otherwise. A *cut-decomposition* of A expresses it in the form

$$A = D^{(1)} + \dots + D^{(s)} + W,$$

where the matrices $D^{(i)}$ are cut matrices, and the matrix $W = (w_{kl})$ has a relatively small cut-norm.

The authors of [8] describe an efficient algorithm that produces, for any given n by m matrix A with entries in $[-1, 1]$, a cut-decomposition in which the number of cut matrices is $O(1/\epsilon^2)$, and the cut-norm of the matrix W is at most ϵnm . This is done as follows. Starting with $W^{(0)} = A$, suppose that the first i cut matrices $D^{(1)}, \dots, D^{(i)}$ have already been defined, and consider the difference $W^{(i)} = A - \sum_{j=1}^i D^{(j)}$. If the cut-norm of this difference is already smaller than ϵnm , we are done. Otherwise, let I, J be sets of rows and columns such that

$$(5.1) \quad \left| \sum_{k \in I, l \in J} w_{kl}^{(i)} \right| \geq \rho \epsilon nm,$$

where $\rho > 0$ is an absolute positive constant. Let d be the average value of the entries $w_{kl}^{(i)}$ for $k \in I, l \in J$, and define $D^{(i+1)} = CUT(I, J, d)$, $W^{(i+1)} = W^{(i)} - D^{(i+1)}$. A simple computation, described in [8], shows that the sum of squares of the entries of the new matrix $W^{(i+1)}$ is at most the sum of squares of the entries of the matrix $W^{(i)}$ minus $\rho^2 \epsilon^2 nm$. Therefore, this process must terminate after at most $\frac{1}{\rho^2 \epsilon^2}$ steps. The main step in the algorithm is clearly that of finding the sets I, J that satisfy (5.1), and the authors are able to do it efficiently only when the cut-norm is at least a constant fraction of nm . Our algorithm here enables us to perform this task efficiently (even if ϵ is, say, $1/n^{0.001}$, which is not feasible using the approach of [8], as the running time of their algorithm is exponential in $1/\epsilon^2$).

Another possible application of our approximation algorithm appears in computational molecular biology. While trying to identify groups of genes with statistically significant correlated behavior across diverse experiments, it is desirable to solve a certain biclustering problem; see [27], [26]. The basic computational problem here is to find efficiently, in a given matrix whose entries are the logarithms of certain probabilities, a submatrix of (approximately) maximum total sum. Our algorithm supplies such a procedure in case the sum of entries in every row (or every column) of the given matrix is zero.

6. Concluding remarks.

- There is a lot known about Grothendieck’s inequality in the case of complex scalars [13], [17], and it may be interesting to find algorithmic or combinatorial applications of these results.
- We believe that the methods described in this paper will find further applications, as they provide a rounding technique that can, at least in the cases considered here, handle cancellations between positive and negative terms.
- It will be interesting to improve the approximation guarantee of the algorithms presented here. It will also be interesting to find a ρ -approximation algorithm for the CUT NORM problem (for any positive absolute constant ρ) which does not apply semidefinite programming and/or can be analyzed without relying on Grothendieck’s inequality.

Acknowledgments. Part of this work was carried out during a visit by the first author to Microsoft Research, Redmond, WA, and he thanks his hosts at Microsoft for their hospitality. The authors also thank L. Babai and M. Goemans for helpful comments.

REFERENCES

- [1] N. ALON, L. BABAI, AND A. ITAI, *A fast and simple randomized parallel algorithm for the maximal independent set problem*, J. Algorithms, 7 (1986), pp. 567–583.
- [2] N. ALON, W. F. DE LA VEGA, R. KANNAN, AND M. KARPINSKI, *Random sampling and approximation of MAX-CSP problems*, in Proceedings of the 34th ACM Symposium on Theory of Computing, ACM Press, New York, 2002, pp. 232–239.
- [3] N. ALON, R. A. DUKE, H. LEFMANN, V. RÖDL, AND R. YUSTER, *The algorithmic aspects of the regularity lemma*, J. Algorithms, 16 (1994), pp. 80–109.
- [4] N. ALON, Y. MATIAS, AND M. SZEGEDY, *The space complexity of approximating the frequency moments*, J. Comput. System Sci., 58 (1999), pp. 137–147.
- [5] N. ALON AND J. SPENCER, *The Probabilistic Method*, 2nd ed., Wiley, New York, 2000.
- [6] S. ARORA, C. LUND, R. MOTWANI, M. SUDAN, AND M. SZEGEDY, *Proof verification and intractability of approximation problems*, J. ACM, 45 (1998), pp. 501–555.
- [7] J. DIESTEL, H. JARCHOW, AND A. TONGE, *Absolutely Summing Operators*, Cambridge University Press, Cambridge, UK, 1995.
- [8] A. M. FRIEZE AND R. KANNAN, *Quick approximation to matrices and applications*, Combinatorica, 19 (1999), pp. 175–200.
- [9] U. FEIGE AND M. LANGBERG, *The RPR² rounding technique for semidefinite programs*, in Proceedings of the 28th International Colloquium on Automata, Languages and Programming, Crete, Greece, 2001, pp. 213–224.
- [10] M. GRÖTSCHEL, L. LOVÁSZ, AND A. SCHRIJVER, *The ellipsoid method and its consequences in combinatorial optimization*, Combinatorica, 1 (1981), pp. 169–197.
- [11] A. GROTHENDIECK, *Résumé de la théorie métrique des produits tensoriels topologiques*, Bol. Soc. Mat. São Paulo, 8 (1953), pp. 1–79.
- [12] M. X. GOEMANS AND D. P. WILLIAMSON, *Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming*, J. ACM, 42 (1995), pp. 1115–1145.
- [13] U. HAAGERUP, *A new upper bound for the complex Grothendieck constant*, Israel J. Math., 60 (1987), pp. 199–224.
- [14] J. HÅSTAD, *Some optimal inapproximability results*, J. ACM, 48 (2001), pp. 798–859.
- [15] G. J. O. JAMESON, *Summing and Nuclear Norms in Banach Space Theory*, London Math. Soc. Stud. Texts 8, Cambridge University Press, Cambridge, UK, 1987.
- [16] W. B. JOHNSON AND J. LINDENSTRAUSS, *Basic concepts in the geometry of Banach spaces*, in Handbook of the Geometry of Banach Spaces, Vol. I, North-Holland, Amsterdam, 2001, pp. 1–84.
- [17] H. KÖNIG, *On the complex Grothendieck constant in the n -dimensional case*, in Geometry of Banach Spaces, London Math. Soc. Lecture Note Ser. 158, P. F. X. Muller and W. Schachermayer, eds., Cambridge University Press, Cambridge, UK, 1990, pp. 181–198.
- [18] J. L. KRIVINE, *Sur la constante de Grothendieck*, C. R. Acad. Sci. Paris Sér. A-B, 284 (1977), pp. 445–446.
- [19] M. LEWIN, D. LIVNAT, AND U. ZWICK, *Improved rounding techniques for the MAX 2-SAT and MAX DI-CUT problems*, in Proceedings of the 9th Conference on Integer Programming and Combinatorial Optimization, Cambridge, MA, 2002, pp. 67–82.
- [20] J. LINDENSTRAUSS AND A. PEŁCZYŃSKI, *Absolutely summing operators in L_p -spaces and their applications*, Studia Math., 29 (1968), pp. 275–326.
- [21] S. MAHAJAN AND H. RAMESH, *Derandomizing approximation algorithms based on semidefinite programming*, SIAM J. Comput., 28 (1999), pp. 1641–1663.
- [22] Y. E. NESTEROV, *Semidefinite relaxation and nonconvex quadratic optimization*, Optim. Methods Softw., 9 (1998), pp. 141–160.
- [23] C. H. PAPADIMITRIOU AND M. YANNAKAKIS, *Optimization, approximation, and complexity classes*, J. Comput. System Sci., 43 (1991), pp. 425–440.
- [24] R. E. RIETZ, *A proof of the Grothendieck inequality*, Israel J. Math., 19 (1974), pp. 271–276.
- [25] E. SZEMERÉDI, *Regular partitions of graphs*, in Problèmes combinatoires et théorie des graphes, Colloq. Internat. CNRS 260, J.-C. Bermond, J.-C. Fournier, M. Las Vergnas, and D. Sotteau, eds., CNRS, Paris, 1978, pp. 399–401.

- [26] A. TANAY, R. SHARAN, M. KUPIEC, AND R. SHAMIR, *Revealing modularity and organization in the yeast molecular network by integrated analysis of highly heterogeneous genome-wide data*, Proc. Nat. Acad. Sci. USA, 101 (2004), pp. 2981–2986.
- [27] A. TANAY, R. SHARAN, AND R. SHAMIR, *Discovering statistically significant biclusters in gene expression data*, Bioinformatics, 18 (2002), pp. 136–144.
- [28] U. ZWICK, *Outward rotations: A tool for rounding solutions of semidefinite programming relaxations, with applications to MAX CUT and other problems*, in Proceedings of the 31st Annual ACM Symposium on Theory of Computing, Atlanta, GA, 1999, pp. 679–687.

LOWER BOUNDS FOR LOCAL SEARCH BY QUANTUM ARGUMENTS*

SCOTT AARONSON†

Abstract. The problem of finding a local minimum of a black-box function is central for understanding local search as well as quantum adiabatic algorithms. For functions on the Boolean hypercube $\{0, 1\}^n$, we show a lower bound of $\Omega(2^{n/4}/n)$ on the number of queries needed by a quantum computer to solve this problem. More surprisingly, our approach, based on Ambainis’s quantum adversary method, also yields a lower bound of $\Omega(2^{n/2}/n^2)$ on the problem’s *classical* randomized query complexity. This improves and simplifies a 1983 result of Aldous. Finally, in both the randomized and quantum cases, we give the first nontrivial lower bounds for finding local minima on grids of constant dimension $d \geq 3$.

Key words. quantum computing, query complexity, decision trees, local search, local optima, polynomial local search, random walks

AMS subject classifications. 60G50, 68Q10, 68Q15, 68Q17, 81P68

DOI. 10.1137/S0097539704447237

1. Introduction. This paper deals with the following problem.

LOCAL SEARCH. *Given an undirected graph $G = (V, E)$ and a function $f : V \rightarrow \mathbb{N}$, find a local minimum of f —that is, a vertex v such that $f(v) \leq f(w)$ for all neighbors w of v .*

We are interested in the number of *queries* that an algorithm needs to solve this problem, where a query just returns $f(v)$ given v . We consider deterministic, randomized, and quantum algorithms. Section 2 motivates the problem theoretically and practically; this section explains our results.

We start with some simple observations. If G is the complete graph of size N , then clearly $\Omega(N)$ queries are needed to find a local minimum (or $\Omega(\sqrt{N})$ with a quantum computer [8]). At the other extreme, if G is a line of length N , then even a deterministic algorithm can find a local minimum in $O(\log N)$ queries, using binary search: query the middle two vertices, v and w . If $f(v) \leq f(w)$, then search the line of length $(N - 2)/2$ connected to v ; otherwise search the line connected to w . Continue recursively in this manner until a local minimum is found.

So the interesting case is when G is a graph of “intermediate” connectedness: for example, the Boolean hypercube $\{0, 1\}^n$, with two vertices adjacent if and only if they have Hamming distance 1. For this graph, Llewellyn, Tovey, and Trick [18, 19] showed an $\Omega(2^n/\sqrt{n})$ lower bound on the number of queries needed by any deterministic algorithm, using a simple adversary argument. Intuitively, until the set of vertices queried so far comprises a *vertex cut* (that is, splits the graph into two or more connected components), an adversary is free to return a descending sequence of f values: $f(v_1) = 2^n$ for the first vertex v_1 queried by the algorithm, $f(v_2) = 2^n - 1$ for the second vertex queried, and so on. Moreover, once the set of queried vertices

*Received by the editors July 12, 2004; accepted for publication (in revised form) July 11, 2005; published electronically February 21, 2006. This work was done while the author was a graduate student at UC Berkeley and during visits to the Hebrew University in Jerusalem and the Perimeter Institute in Waterloo. The research was supported by an NSF Graduate Fellowship, by NSF ITR grant CCR-0121555, and by the Defense Advanced Research Projects Agency (DARPA).

<http://www.siam.org/journals/sicomp/35-4/44723.html>

†University of Waterloo, Waterloo, ON, N2L 3G1, Canada (scott@scottaaronson.com).

does comprise a cut, the adversary can choose the largest connected component of unqueried vertices and restrict the problem recursively to that component. So to lower-bound the deterministic query complexity, it suffices to lower-bound the size of any cut that splits the graph into two reasonably large components.¹ For the Boolean hypercube, Llewellyn, Tovey, and Trick showed that the best one can do is essentially to query all $\Omega(2^n/\sqrt{n})$ vertices of Hamming weight $n/2$.

The argument of Llewellyn, Tovey, and Trick fails completely in the case of randomized algorithms. By Yao's minimax principle, what we want here is a fixed *distribution* \mathcal{D} over functions $f : \{0, 1\}^n \rightarrow \mathbb{N}$, such that any deterministic algorithm needs many queries to find a local minimum of f , with high probability if f is drawn from \mathcal{D} . Taking \mathcal{D} to be uniform will not do, since a local minimum of a uniform random function is easily found. However, Aldous [3] had the idea of defining \mathcal{D} via a *random walk*, as follows. Choose a vertex $v_0 \in \{0, 1\}^n$ uniformly at random; then perform an unbiased walk² v_0, v_1, v_2, \dots starting from v_0 . For each vertex v , set $f(v)$ equal to the first hitting time of the walk at v —that is, $f(v) = \min\{t : v_t = v\}$. Clearly any f produced in this way has a unique local minimum at v_0 , since for all $t > 0$, if vertex v_t is visited for the first time at step t , then $f(v_t) > f(v_{t-1})$. Using sophisticated random walk analysis, Aldous managed to show a lower bound of $2^{n/2-o(n)}$ on the expected number of queries needed by any randomized algorithm to find v_0 .³ (As we will see in section 3, this lower bound is close to tight.) Intuitively, since a random walk on the hypercube mixes in $O(n \log n)$ steps, an algorithm that has not queried a v with $f(v) < 2^{n/2}$ has almost no useful information about where the unique minimum v_0 is, so its next query will just be a “stab in the dark.”

However, Aldous's result leaves several questions about LOCAL SEARCH unanswered. What if the graph G is a three-dimensional (3-D) cube, on which a random walk does *not* mix very rapidly? Can we still lower-bound the randomized query complexity of finding a local minimum? More generally, what parameters of G make the problem hard or easy? Also, what is the quantum query complexity of LOCAL SEARCH?

This paper presents a new approach to LOCAL SEARCH, which we believe points the way to a complete understanding of its complexity. Our approach is based on the *quantum adversary method*, introduced by Ambainis [4] to prove lower bounds on quantum query complexity. Surprisingly, our approach yields new and simpler lower bounds for the problem's *classical* randomized query complexity, in addition to quantum lower bounds. Thus, along with recent work by Kerenidis and de Wolf [15] and Aharonov and Regev [2] among others, this paper illustrates how quantum ideas can help to resolve classical open problems.

Our results are as follows. For the Boolean hypercube $G = \{0, 1\}^n$, we show that any quantum algorithm needs $\Omega(2^{n/4}/n)$ queries to find a local minimum on G , and any randomized algorithm needs $\Omega(2^{n/2}/n^2)$ queries (improving the $2^{n/2-o(n)}$ lower bound of Aldous [3]). Our proofs are elementary and do not require random walk analysis. By comparison, the best known upper bounds are $O(2^{n/3}n^{1/6})$ for a quantum algorithm and $O(2^{n/2}\sqrt{n})$ for a randomized algorithm. If G is a d -

¹Llewellyn, Tovey, and Trick actually give a tight characterization of deterministic query complexity in terms of vertex cuts.

²Actually, Aldous used a continuous-time random walk, so the functions would be from $\{0, 1\}^n$ to \mathbb{R} .

³Independently and much later, Droste, Jansen, and Wegener [11] showed the weaker bound $2^{g(n)}$ for any $g(n) = o(n)$.

dimensional grid of size $N^{1/d} \times \dots \times N^{1/d}$, where $d \geq 3$ is a constant, then we show that any quantum algorithm needs $\Omega(\sqrt{N^{1/2-1/d}/\log N})$ queries to find a local minimum on G , and any randomized algorithm needs $\Omega(N^{1/2-1/d}/\log N)$ queries. No nontrivial lower bounds (randomized or quantum) were previously known in this case.⁴ For any graph with N vertices and degree δ , we also show an upper bound of $O(N^{1/3}\delta^{1/6})$ on the quantum query complexity.

The paper is organized as follows. Section 2 motivates lower bounds on LOCAL SEARCH, pointing out connections to simulated annealing, quantum adiabatic algorithms, and the complexity class TFNP of total function problems. Section 3 defines notation and reviews basic facts about LOCAL SEARCH, including the $O(N^{1/3}\delta^{1/6})$ upper bound. In section 4 we give an intuitive explanation of Ambainis’s quantum adversary method, then state and prove a classical analogue of Ambainis’s main lower bound theorem. Section 5 introduces *snakes*, a construction by which we apply the two adversary methods to LOCAL SEARCH. We show there that to prove lower bounds for any graph G , it suffices to upper-bound a combinatorial parameter ε of a “snake distribution” on G . Section 6 applies this framework to specific examples of graphs: the Boolean hypercube in section 6.1 and the d -dimensional grid in section 6.2.

1.1. Current status. In an earlier version of this paper, we raised as our “most ambitious” conjecture that the deterministic and quantum query complexities of LOCAL SEARCH are polynomially related for *every* family of graphs. At the time, it was not even known whether deterministic and *randomized* query complexities were polynomially related, not even for simple examples such as the two-dimensional (2-D) square grid. Subsequently Santha and Szegedy [23] spectacularly resolved our conjecture, by showing that for every family of graphs, the quantum query complexity of LOCAL SEARCH is at least the 19th root (!) of the deterministic query complexity. They also showed that the quantum query complexity is $\Omega(N^{1/8})$ for the 2-D square grid.

So, like the King in the story of Rumpelstiltskin, we now feel emboldened to make a stronger conjecture: *for every family of graphs, the randomized query complexity of LOCAL SEARCH is at least the square root of the deterministic query complexity, the quantum query complexity is at least the square root of the randomized query complexity, and the quantum query complexity is at least the cube root of the deterministic query complexity.* Intuitively, there is “never anything better to do” than some combination of Grover’s algorithm with the classical steepest descent and divide-and-conquer heuristics.

Recent progress on specific graph families lends some support to our conjecture. For the Boolean hypercube $\{0, 1\}^n$, Zhang [27] has shown that the known algorithms are exactly optimal—that is, the randomized query complexity of LOCAL SEARCH is $\Theta(2^{n/2}\sqrt{n})$, and the quantum query complexity is $\Theta(2^{n/3}n^{1/6})$. Zhang has also shown the following lower bounds for the d -dimensional grid of size $N^{1/d} \times \dots \times N^{1/d}$:

Dimension	Randomized	Quantum
$d = 2$	$\Omega(N^{1/3})$	$\Omega(N^{1/6})$
$d = 3$	$\Omega\left((N/\log N)^{1/2}\right)$	$\Omega(N^{1/4})$
$d = 4$	$\Omega(N^{1/2})$	$\Omega(N^{3/10})$
$d = 5$	$\Omega(N^{1/2})$	$\Omega\left((N/\log N)^{1/3}\right)$
$d \geq 6$	$\Omega(N^{1/2})$	$\Omega(N^{1/3})$

⁴A lower bound on deterministic query complexity was known for such graphs [17].

Note that Zhang’s randomized lower bound is tight when $d \geq 4$, and his quantum lower bound is tight when $d \geq 6$. Finally, in the $d = 2$ case, Zhang has given a quantum upper bound of $O(N^{1/4} (\log \log N)^2)$, which Verhoeven [25] very recently improved to $O(N^{1/4} \log \log N)$ and generalized to all planar and bounded-genus graphs. Currently, no randomized upper bound better than $O(\sqrt{N})$ is known when $d \geq 2$, and no quantum upper bound better than $O(N^{1/3})$ is known when $d \geq 3$.

In a different direction, Laplante and Magniez [16] proposed a variant of Ambainis’s adversary method based on Kolmogorov complexity and showed that their variant also allows randomized and quantum query complexity to be lower-bounded in a unified way. Subsequently, Špalek and Szegedy [26] showed that Laplante and Magniez’s variant is equivalent to a weighted generalization of Ambainis’s original adversary method.

2. Motivation. Local search is the most effective weapon ever devised against hard optimization problems. For many real applications, neither backtrack search, nor approximation algorithms, nor even Grover’s algorithm (assuming we had a quantum computer) can compare. Furthermore, along with quantum computing, local search (broadly defined) is one of the most interesting links between computer science and Nature. It is related to evolutionary biology via genetic algorithms and to the physics of materials via simulated annealing. Thus it is both practically and scientifically important to understand its performance.

The conventional wisdom is that, although local search performs well in practice, its central (indeed defining) flaw is a tendency to get stuck at local optima. If this were correct, one corollary would be that the reason local search performs so well is that the problem it really solves—finding a local optimum—is intrinsically easy. It would thus be unnecessary to seek further explanations for its performance. Another corollary would be that, for *unimodal* functions (which have no local optima besides the global optimum), the global optimum would be easily found.

However, the conventional wisdom is false. The results of Llewellyn, Tovey, and Trick [18, 19] and Aldous [3] show that even if f is unimodal, any classical algorithm that treats f as a black box needs exponential time to find the global minimum of f in general. Our results extend this conclusion to quantum algorithms. In our view, the practical upshot of these results is that they force us to confront the question, what is it about “real-world” problems that makes it easy to find a local optimum? That is, why do exponentially long chains of descending values, such as those used for lower bounds, almost never occur in practice (even in functions with large range sizes)? One possibility is that the functions that occur in practice look “globally” like random functions, but we do not know whether that is true in any meaningful sense.

Our results are also relevant for physics. Many physical systems, including folding proteins and networks of springs and pulleys, can be understood as performing “local search” through an energy landscape to reach a locally minimal energy configuration. A key question is, how long will the system take to reach its ground state (that is, a globally minimal configuration)? Of course, if there are local optima, the system might *never* reach its ground state, just as a rock in a mountain crevice does not roll to the bottom by going up first. But what if the energy landscape is unimodal? And moreover, what if the physical system is quantum? Our results show that, for certain energy landscapes, even a quantum system would take exponential time to reach its ground state, regardless of what Hamiltonian is applied to it. So, in particular, the quantum adiabatic algorithm proposed by Farhi et al. [13], which can be seen as a quantum analogue of simulated annealing, needs exponential time to find a local

minimum in the worst case.

Finally, our results have implications for so-called *total function problems* in complexity theory. Megiddo and Papadimitriou [20] defined a complexity class⁵ TFNP, consisting (informally) of those NP search problems for which a solution always exists. For example, we might be given a function $f : \{0, 1\}^n \rightarrow \{0, 1\}^{n-1}$ as a Boolean circuit and asked to find any distinct x, y pair such that $f(x) = f(y)$. This particular problem belongs to a subclass of TFNP called PPP (Polynomial Pigeonhole Principle). Notice that no promise is involved: the combinatorial nature of the problem itself forces a solution to exist, even if we have no idea how to find it. In a recent talk, Papadimitriou [22] asked broadly whether such “nonconstructive existence problems” might be good candidates for efficient quantum algorithms. In the case of PPP problems, the collision lower bound of Aaronson [1] (improved by Shi [24] and others) implies a negative answer in the black-box setting. For other subclasses of TFNP, such as PODN (polynomial odd-degree node), a quantum black-box lower bound follows easily from the optimality of Grover’s search algorithm.

However, there is one important subclass of TFNP for which no quantum lower bound was previously known. This is PLS (polynomial local search), defined by Johnson, Papadimitriou, and Yannakakis [14] as a class of optimization problems whose cost function f and neighborhood function η (that is, the set of neighbors of a given point) are both computable in polynomial time. Given such a problem, the task is to output any local minimum of the cost function: that is, a v such that $f(v) \leq f(w)$ for all $w \in \eta(v)$. The lower bound of Llewellyn, Tovey, and Trick [18, 19] yields an oracle A relative to which $\text{FP}^A \neq \text{PLS}^A$, by a standard diagonalization argument along the lines of Baker, Gill, and Solovay [6]. Likewise, the lower bound of Aldous [3] yields an oracle relative to which $\text{PLS} \not\subseteq \text{FBPP}$, where FBPP is simply the function version of BPP. Our results yield the first oracle relative to which $\text{PLS} \not\subseteq \text{FBQP}$. In light of this oracle separation, we raise an admittedly vague question, is there a nontrivial “combinatorial” subclass of TFNP that we can show *is* contained in FBQP?

3. Preliminaries. In the LOCAL SEARCH problem, we are given an undirected graph $G = (V, E)$ with $N = |V|$ and oracle access to a function $f : V \rightarrow \mathbb{N}$. The goal is to find any *local minimum* of f , defined as a vertex $v \in V$ such that $f(v) \leq f(w)$ for all neighbors w of v . Clearly such a local minimum exists. We want to find one using as few queries as possible, where a query returns $f(v)$ given v . Queries can be adaptive; that is, they can depend on the outcomes of previous queries. We assume G is known in advance, so that only f needs to be queried. Since we care only about query complexity, not computation time, there is no difficulty in dealing with an infinite range for f —though for our lower bounds, it will turn out that a range of size $O(|V|)$ suffices.

Our model of query algorithms is the standard one; see [9] for a survey. Given a graph G , the deterministic query complexity of LOCAL SEARCH on G , which we denote $\text{DLS}(G)$, is $\min_{\Gamma} \max_f T(\Gamma, f, G)$ where the minimum ranges over all deterministic algorithms Γ , the maximum ranges over all f , and $T(\Gamma, f, G)$ is the number of queries made to f by Γ before it halts and outputs a local minimum of f (or ∞ if Γ fails to do so). The randomized query complexity $\text{RLS}(G)$ is defined similarly, except that now the algorithm has access to an infinite random string R and must only output a local minimum with probability at least $2/3$ over R . For simplicity, we assume that the number of queries T is the same for all R ; clearly this assumption changes the

⁵See www.complexityzoo.com for details about the complexity classes mentioned in this paper.

complexity by at most a constant factor.

In the quantum model, an algorithm's state has the form $\sum_{v,z,s} \alpha_{v,z,s} |v, z, s\rangle$, where v is the label of a vertex in G , and z and s are strings representing the answer register and workspace, respectively. The $\alpha_{v,z,s}$'s are complex amplitudes satisfying $\sum_{v,z,s} |\alpha_{v,z,s}|^2 = 1$. Starting from an arbitrary (fixed) initial state, the algorithm proceeds by an alternating sequence of *queries* and *algorithm steps*. A query maps each $|v, z, s\rangle$ to $|v, z \oplus f(v), s\rangle$, where \oplus denotes bitwise exclusive-OR. An algorithm step multiplies the vector of $\alpha_{v,z,s}$'s by an arbitrary unitary matrix that does not depend on f . Letting \mathcal{M}_f denote the set of local minima of f , the algorithm succeeds if at the end $\sum_{v,z,s : v \in \mathcal{M}_f} |\alpha_{v,z,s}|^2 \geq \frac{2}{3}$. Then the bounded-error quantum query complexity, or $\text{QLS}(G)$, is defined as the minimum number of queries used by a quantum algorithm that succeeds on every f .

It is immediate that $\text{QLS}(G) \leq \text{RLS}(G) \leq \text{DLS}(G) \leq N$. Also, letting δ be the maximum degree of G , we have the following trivial lower bound.

PROPOSITION 3.1. $\text{RLS}(G) = \Omega(\delta)$ and $\text{QLS}(G) = \Omega(\sqrt{\delta})$.

Proof. Let v be a vertex of G with degree δ . Choose a neighbor w of v uniformly at random, and let $f(w) = 1$. Let $f(v) = 2$, and $f(u) = 3$ for all neighbors u of v other than w . Let S be the neighbor set of v (including v itself); then for all $x \notin S$, let $f(x) = 3 + \Delta(x, S)$, where $\Delta(x, S)$ is the minimum distance from x to a vertex in S . Clearly f has a unique local minimum at w . However, finding y requires exhaustive search among the δ neighbors of v , which takes $\Omega(\sqrt{\delta})$ quantum queries by Bennett et al. [8]. \square

A corollary of Proposition 3.1 is that classically, zero-error randomized query complexity is equivalent to bounded-error up to a constant factor. For given a candidate local minimum v , one can check using $O(\delta)$ queries that v is indeed a local minimum. Since $\Omega(\delta)$ queries are needed anyway, this verification step does not affect the overall complexity.

As pointed out by Aldous [3], a classical randomized algorithm can find a local minimum of f with high probability in $O(\sqrt{N\delta})$ queries. The algorithm just queries $\sqrt{N\delta}$ vertices uniformly at random, and lets v_0 be a queried vertex for which $f(v)$ is minimal. It then follows v_0 to a local minimum by steepest descent. That is, for $t = 0, 1, 2, \dots$, it queries all neighbors of v_t , halts if v_t is a local minimum, and otherwise sets v_{t+1} to be the neighbor w of v_t for which $f(w)$ is minimal (breaking ties by lexicographic ordering). A similar idea yields an improved quantum upper bound.

THEOREM 3.2. For any G , $\text{QLS}(G) = O(N^{1/3}\delta^{1/6})$.

Proof. The algorithm first chooses $N^{2/3}\delta^{1/3}$ vertices of G uniformly at random, then uses Grover search to find a chosen vertex v_0 for which $f(v)$ is minimal. By a result of Dürr and Høyer [12], this can be done with high probability in $O(N^{1/3}\delta^{1/6})$ queries. Next, for $t = 0, 1, 2, \dots$, the algorithm performs Grover search over all neighbors of v_t , looking for a neighbor w such that $f(w) < f(v_t)$. If it finds such a w , then it sets $v_{t+1} := w$ and continues to the next iteration. Otherwise, it repeats the Grover search $\log(N/\delta)$ times before finally giving up and returning v_t as a claimed local minimum.

The expected number of u such that $f(u) < f(v_0)$ is at most $N / (N^{2/3}\delta^{1/3}) = (N/\delta)^{1/3}$. Since $f(v_{t+1}) < f(v_t)$ for all t , clearly the number of such u provides an upper bound on t . Furthermore, assuming there exists a w such that $f(w) < f(v_t)$, the expected number of repetitions of Grover's algorithm until such a w is found is $O(1)$. Since each repetition takes $O(\sqrt{\delta})$ queries, by linearity of expectation the total

expected number of queries used by the algorithm is therefore

$$O\left(N^{1/3}\delta^{1/6} + (N/\delta)^{1/3}\sqrt{\delta} + \log(N/\delta)\sqrt{\delta}\right)$$

or $O(N^{1/3}\delta^{1/6})$. To see that the algorithm finds a local minimum with high probability, observe that for each t the probability of not finding a w such that $f(w) < f(v_t)$, given that one exists, is at most $c^{-\log(N/\delta)} \leq (\delta/N)^{1/3}/10$ for a suitable constant c . So by the union bound, the probability that the algorithm returns a “false positive” is at most $(N/\delta)^{1/3} \cdot (\delta/N)^{1/3}/10 = 1/10$. \square

4. Relational adversary method. We know essentially two methods for proving lower bounds on quantum query complexity: the polynomial method of Beals et al. [7], and the quantum adversary method of Ambainis [4].⁶ For a few problems, such as the collision problem [1, 24], the polynomial method succeeded where the adversary method failed. However, for problems that lack permutation symmetry (such as LOCAL SEARCH), the adversary method has proven more effective.⁷

How could a quantum lower bound method possibly be applied classically? When proving randomized lower bounds, the tendency is to attack “bare-handed”: fix a distribution over inputs, and let x_1, \dots, x_t be the locations queried so far by the algorithm. Show that for small t , the posterior distribution over inputs, *conditioned* on x_1, \dots, x_t , is still “hard” with high probability—so that the algorithm knows almost nothing even about which location x_{t+1} to query next. This is essentially the approach taken by Aldous [3] to prove a $2^{n/2-o(n)}$ lower bound on RLS $(\{0, 1\}^n)$.

In the quantum case, however, it is unclear how to specify what an algorithm “knows” after a given number of queries. So we are almost *forced* to step back and identify general combinatorial properties of input sets that make them hard to distinguish. Once we have such properties, we can then try to exhibit them in functions of interest.

We will see, somewhat surprisingly, that this “gloved” approach is useful for classical lower bounds as well as quantum ones. In our *relational adversary method*, we assume there exists a T -query randomized algorithm for function F . We consider a set \mathcal{A} of 0-inputs of F , a set \mathcal{B} of 1-inputs, and an arbitrary real-valued *relation function* $R(A, B) \geq 0$ for $A \in \mathcal{A}$ and $B \in \mathcal{B}$. Intuitively, $R(A, B)$ should be large if A and B differ in only a few locations. We then fix a probability distribution \mathcal{D} over inputs; by Yao’s minimax principle, there exists a T -query deterministic algorithm Γ^* that succeeds with high probability on inputs drawn from \mathcal{D} . Let W_A be the set of 0-inputs and W_B the set of 1-inputs on which Γ^* succeeds. Using the relation function R , we define a *separation measure* S between W_A and W_B and show that (1) initially $S = 0$, (2) by the end of the computation S must be large, and (3) S increases by only a small amount as the result of each query. It follows that T must be large.

The advantage of the relational method is that it converts a “dynamic” opponent—an algorithm that queries adaptively—into a relatively static one. It thereby makes it easier to focus on what is unique about a problem and ignore aspects of query complexity that are common to all problems. Furthermore, one does not need to know anything about quantum computing to understand and apply the method. On the other hand, it is not clear how one would come up with it in the first place, without

⁶We are thinking here of the hybrid method [8] as a cousin of the adversary method.

⁷Indeed, Ambainis [5] has given problems for which the adversary method provably yields a better lower bound than the polynomial method.

Ambainis’s *quantum* adversary method [4] and the reasoning about entanglement that led to it.

Our starting point is the “most general” adversary theorem in Ambainis’s original paper (Theorem 6 in [4]), which he introduced to prove a quantum lower bound for the problem of inverting a permutation. Here the input is a permutation $\sigma(1), \dots, \sigma(N)$, and the task is to output 0 if $\sigma^{-1}(1) \leq N/2$ and 1 otherwise. To lower-bound this problem’s query complexity, what we would like to say is the following.

Given any 0-input σ and any location x , if we choose a random 1-input τ that is “related” to σ , then the probability $\theta(\sigma, x)$ over τ that $\sigma(x)$ does not equal $\tau(x)$ is small. In other words, the algorithm is unlikely to distinguish σ from a random neighbor τ of σ by querying x .

Unfortunately, the above claim is false. Letting $x = \sigma^{-1}(1)$, we have that $\sigma(x) \neq \tau(x)$ for every 1-input τ , and thus $\theta(\sigma, x) = 1$. Ambainis resolves this difficulty by letting us take the maximum, over all 0-inputs σ and 1-inputs τ that are related and differ at x , of the *geometric mean* $\sqrt{\theta(\sigma, x)\theta(\tau, x)}$. Even if $\theta(\sigma, x) = 1$, the geometric mean is still small provided that $\theta(\tau, x)$ is small. More formally, we have the following theorem.

THEOREM 4.1 (Ambainis). *Let $\mathcal{A} \subseteq F^{-1}(0)$ and $\mathcal{B} \subseteq F^{-1}(1)$ be sets of inputs to function F . Let $R(A, B) \geq 0$ be a real-valued function, and for $A \in \mathcal{A}$, $B \in \mathcal{B}$, and location x , let*

$$\theta(A, x) = \frac{\sum_{B^* \in \mathcal{B} : A(x) \neq B^*(x)} R(A, B^*)}{\sum_{B^* \in \mathcal{B}} R(A, B^*)},$$

$$\theta(B, x) = \frac{\sum_{A^* \in \mathcal{A} : A^*(x) \neq B(x)} R(A^*, B)}{\sum_{A^* \in \mathcal{A}} R(A^*, B)},$$

where the denominators are all nonzero. Then the number of quantum queries needed to evaluate F with at least 9/10 probability is $\Omega(1/v_{\text{geom}})$, where

$$v_{\text{geom}} = \max_{A \in \mathcal{A}, B \in \mathcal{B}, x : R(A, B) > 0, A(x) \neq B(x)} \sqrt{\theta(A, x)\theta(B, x)}.$$

The best way to understand Theorem 4.1 is to see it used in an example.

PROPOSITION 4.2 (Ambainis). *The quantum query complexity of inverting a permutation is $\Omega(\sqrt{N})$.*

Proof. Let \mathcal{A} be the set of all permutations σ such that $\sigma^{-1}(1) \leq N/2$, and \mathcal{B} be the set of permutations τ such that $\tau^{-1}(1) > N/2$. Given $\sigma \in \mathcal{A}$ and $\tau \in \mathcal{B}$, let $R(\sigma, \tau) = 1$ if σ and τ differ only at locations $\sigma^{-1}(1)$ and $\tau^{-1}(1)$, and $R(\sigma, \tau) = 0$ otherwise. Then given σ, τ with $R(\sigma, \tau) = 1$, if $x \leq N/2$ then $\theta(\tau, x) \leq 2/N$, and if $x > N/2$ then $\theta(\sigma, x) \leq 2/N$. So $\max_x : \sigma(x) \neq \tau(x) \sqrt{\theta(\sigma, x)\theta(\tau, x)} \leq \sqrt{2/N}$. \square

The only difference between Theorem 4.1 and our relational adversary theorem is that in the latter, we take the *minimum* of $\theta(A, x)$ and $\theta(B, x)$ instead of the geometric mean. Taking the reciprocal then gives up to a quadratically better lower bound: for example, we obtain that the randomized query complexity of inverting a permutation is $\Omega(N)$. However, the proofs of the two theorems are quite different.

THEOREM 4.3. *Let $\mathcal{A}, \mathcal{B}, R, \theta$ be as in Theorem 4.1. Then the number of randomized queries needed to evaluate F with at least 9/10 probability is $\Omega(1/v_{\text{min}})$, where*

$$v_{\text{min}} = \max_{A \in \mathcal{A}, B \in \mathcal{B}, x : R(A, B) > 0, A(x) \neq B(x)} \min\{\theta(A, x), \theta(B, x)\}.$$

Proof. Let Γ be a randomized algorithm that, given an input A , returns $F(A)$ with at least $9/10$ probability. Let T be the number of queries made by Γ . For all $A \in \mathcal{A}$, $B \in \mathcal{B}$, define

$$\begin{aligned} M(A) &= \sum_{B^* \in \mathcal{B}} R(A, B^*), \\ M(B) &= \sum_{A^* \in \mathcal{A}} R(A^*, B), \\ M &= \sum_{A^* \in \mathcal{A}} M(A^*) = \sum_{B^* \in \mathcal{B}} M(B^*). \end{aligned}$$

Now let \mathcal{D}_A be the distribution over $A \in \mathcal{A}$ in which each A is chosen with probability $M(A)/M$; and let \mathcal{D}_B be the distribution over $B \in \mathcal{B}$ in which each B is chosen with probability $M(B)/M$. Let \mathcal{D} be an equal mixture of \mathcal{D}_A and \mathcal{D}_B . By Yao's minimax principle, there exists a deterministic algorithm Γ^* that makes T queries and succeeds with at least $9/10$ probability given an input drawn from \mathcal{D} . Therefore Γ^* succeeds with at least $4/5$ probability given an input drawn from \mathcal{D}_A alone or from \mathcal{D}_B alone. In other words, letting W_A be the set of $A \in \mathcal{A}$ and W_B the set of $B \in \mathcal{B}$ on which Γ^* succeeds, we have

$$\sum_{A \in W_A} M(A) \geq \frac{4}{5}M, \quad \sum_{B \in W_B} M(B) \geq \frac{4}{5}M.$$

Define a predicate $P^{(t)}(A, B)$, which is true if Γ^* has distinguished $A \in \mathcal{A}$ from $B \in \mathcal{B}$ by the t th query and false otherwise. (To distinguish A from B means to query an index x for which $A(x) \neq B(x)$, given either A or B as input.) Also, for all $A \in \mathcal{A}$, define a score function

$$S^{(t)}(A) = \sum_{B^* \in \mathcal{B} : P^{(t)}(A, B^*)} R(A, B^*).$$

This function measures how much "progress" has been made so far in separating A from \mathcal{B} inputs, where the \mathcal{B} inputs are weighted by $R(A, B)$. Similarly, for all $B \in \mathcal{B}$ define

$$S^{(t)}(B) = \sum_{A^* \in \mathcal{A} : P^{(t)}(A^*, B)} R(A^*, B).$$

It is clear that for all t ,

$$\sum_{A \in \mathcal{A}} S^{(t)}(A) = \sum_{B \in \mathcal{B}} S^{(t)}(B).$$

So we can denote the above sum by $S^{(t)}$ and think of it as a global progress measure. We will show the following about $S^{(t)}$:

- (i) $S^{(0)} = 0$ initially.
- (ii) $S^{(T)} \geq 3M/5$ by the end.
- (iii) $\Delta S^{(t)} \leq 3v_{\min}M$ for all t , where $\Delta S^{(t)} = S^{(t)} - S^{(t-1)}$ is the amount by which $S^{(t)}$ increases as the result of a single query.

It follows from (i)–(iii) that

$$T \geq \frac{3M/5}{3v_{\min}M} = \frac{1}{5v_{\min}},$$

which establishes the theorem. Part (i) is obvious. For part (ii), observe that for every pair (A, B) with $A \in W_A$ and $B \in W_B$, the algorithm Γ^* must query an x such that $A(x) \neq B(x)$. Thus

$$\begin{aligned} S^{(T)} &\geq \sum_{A \in W_A, B \in W_B} R(A, B) \\ &\geq \sum_{A \in W_A} M(A) - \sum_{B \notin W_B} M(B) \\ &\geq \frac{4}{5}M - \frac{1}{5}M. \end{aligned}$$

It remains only to show part (iii). Suppose $\Delta S^{(t)} > 3v_{\min}M$ for some t ; we will obtain a contradiction. Let

$$\Delta S^{(t)}(A) = S^{(t)}(A) - S^{(t-1)}(A),$$

and let C_A be the set of $A \in \mathcal{A}$ for which $\Delta S^{(t)}(A) > v_{\min}M(A)$. Since

$$\sum_{A \in \mathcal{A}} \Delta S^{(t)}(A) = \Delta S^{(t)} > 3v_{\min}M,$$

it follows by Markov's inequality that

$$\sum_{A \in C_A} \Delta S^{(t)}(A) \geq \frac{2}{3} \Delta S^{(t)}.$$

Similarly, if we let C_B be the set of $B \in \mathcal{B}$ for which $\Delta S^{(t)}(B) > v_{\min}M(B)$, we have

$$\sum_{B \in C_B} \Delta S^{(t)}(B) \geq \frac{2}{3} \Delta S^{(t)}.$$

In other words, at least $2/3$ of the increase in $S^{(t)}$ comes from (A, B) pairs such that $A \in C_A$, and at least $2/3$ comes from (A, B) pairs such that $B \in C_B$. Hence, by a ‘‘pigeonhole’’ argument, there exists an $A \in C_A$ and $B \in C_B$ with $R(A, B) > 0$ that are distinguished by the t th query. In other words, there exists an x with $A(x) \neq B(x)$, such that the t th index queried by Γ^* is x whether the input is A or B . Then since $A \in C_A$, we have $v_{\min}M(A) < \Delta S^{(t)}(A)$, and hence

$$\begin{aligned} v_{\min} &< \frac{\Delta S^{(t)}(A)}{M(A)} \\ &\leq \frac{\sum_{B^* \in \mathcal{B} : A(x) \neq B^*(x)} R(A, B^*)}{\sum_{B^* \in \mathcal{B}} R(A, B^*)}, \end{aligned}$$

which equals $\theta(A, x)$. Similarly $v_{\min} < \theta(B, x)$ since $B \in C_B$. This contradicts the definition

$$v_{\min} = \max_{A \in \mathcal{A}, B \in \mathcal{B}, x : R(A, B) > 0, A(x) \neq B(x)} \min \{ \theta(A, x), \theta(B, x) \},$$

and we are done. \square

5. Snakes. For our lower bounds, it will be convenient to generalize random walks to arbitrary distributions over paths, which we call *snakes*.

DEFINITION 5.1. *Given a vertex h in G and a positive integer L , a snake distribution $\mathcal{D}_{h,L}$ (parameterized by h and L) is a probability distribution over paths (x_0, \dots, x_{L-1}) in G , such that each x_t is either equal or adjacent to x_{t+1} , and $x_{L-1} = h$. Let $D_{h,L}$ be the support of $\mathcal{D}_{h,L}$. Then an element of $D_{h,L}$ is called a snake; the part near $x_{L-1} = h$ is the head and the part near x_0 is the tail.*

Given a snake X and integer t , we use $X[t]$ as shorthand for $\{x_0, \dots, x_t\}$.

DEFINITION 5.2. *We say a snake $X \in D_{h,L}$ is ε -good if the following holds. Choose j uniformly at random from $\{0, \dots, L-1\}$, and let $Y = (y_0, \dots, y_{L-1})$ be a snake drawn from $\mathcal{D}_{h,L}$ conditioned on $x_t = y_t$ for all $t > j$. Then*

- (i) *letting $S_{X,Y}$ be the set of vertices v in $X \cap Y$ such that $\min\{t : x_t = v\} = \min\{t : y_t = v\}$, we have*

$$\Pr_{j,Y} [X \cap Y = S_{X,Y}] \geq \frac{9}{10};$$

- (ii) *for all vertices v ,*

$$\Pr_{j,Y} [v \in Y[j]] \leq \varepsilon.$$

The procedure above—wherein we choose a j uniformly at random and then draw a Y from $\mathcal{D}_{h,L}$ consistent with X on all steps later than j —will be important in what follows. We call it *the snake X flicking its tail*. Intuitively, a snake is good if it is spread out fairly evenly in G —so that when it flicks its tail, (1) with high probability the old and new tails do not intersect, and (2) any particular vertex is hit by the new tail with probability at most ε .

We now explain our “snake method” for proving lower bounds for LOCAL SEARCH. Given a snake X , we define an input f_X with a unique local minimum at x_0 , and f values that decrease along X from head to tail. Then, given inputs f_X and f_Y with $X \cap Y = S_{X,Y}$, we let the relation function $R(f_X, f_Y)$ be proportional to the probability that snake Y is obtained by X flicking its tail. (If $X \cap Y \neq S_{X,Y}$ we let $R = 0$.) Let f_X and g_Y be inputs with $R(f_X, g_Y) > 0$, and let v be a vertex such that $f_X(v) \neq g_Y(v)$. Then if all snakes were good, there would be two mutually exclusive cases: (1) v belongs to the tail of X or (2) v belongs to the tail of Y . In case (1), v is hit with small probability when Y flicks its tail, so $\theta(f_Y, v)$ is small. In case (2), v is hit with small probability when X flicks its tail, so $\theta(f_X, v)$ is small. In either case, then, the *geometric mean* $\sqrt{\theta(f_X, v)\theta(f_Y, v)}$ and *minimum* $\min\{\theta(f_X, v), \theta(f_Y, v)\}$ are small. So even though $\theta(f_X, v)$ or $\theta(f_Y, v)$ could be large individually, Theorems 4.1 and 4.3 yield a good lower bound, as in the case of inverting a permutation (see Figure 1).

One difficulty is that not all snakes are good; at best, a large fraction of them are. We could try deleting all inputs f_X such that X is not good, but that might ruin some remaining inputs, which would then have fewer neighbors. So we would have to delete *those* inputs as well, and so on ad infinitum. What we need is basically a way to replace “all inputs” by “most inputs” in Theorems 4.1 and 4.3.

Fortunately, a simple graph-theoretic lemma can accomplish this. The lemma (see Diestel [10, p. 6], for example) says that any graph with average degree at least k contains an induced subgraph with *minimum* degree at least $k/2$. Here we prove a weighted analogue of the lemma.

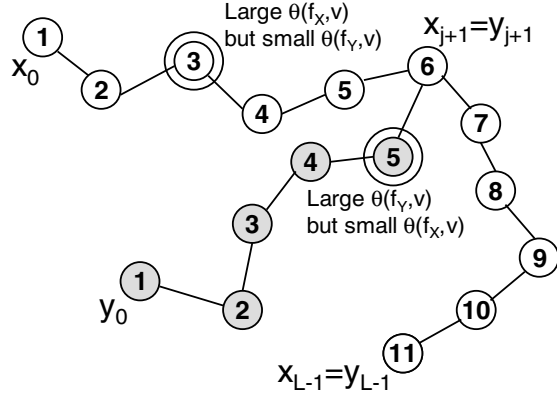


FIG. 1. For every vertex v such that $f_X(v) \neq f_Y(v)$, either when snake X flicks its tail v is not hit with high probability, or when snake Y flicks its tail v is not hit with high probability.

LEMMA 5.3. Let $p(1), \dots, p(m)$ be positive reals summing to 1. Also let $w(i, j)$ for $i, j \in \{1, \dots, m\}$ be nonnegative reals satisfying $w(i, j) = w(j, i)$ and $\sum_{i,j} w(i, j) \geq r$. Then there exists a nonempty subset $U \subseteq \{1, \dots, m\}$ such that for all $i \in U$, $\sum_{j \in U} w(i, j) \geq rp(i)/2$.

Proof. If $r = 0$, then the lemma trivially holds, so assume $r > 0$. We construct U via an iterative procedure. Let $U(0) = \{1, \dots, m\}$. Then for all t , if there exists an $i^* \in U(t)$ for which

$$\sum_{j \in U(t)} w(i^*, j) < \frac{r}{2} p(i^*),$$

then set $U(t+1) = U(t) \setminus \{i^*\}$. Otherwise halt and return $U = U(t)$. To see that the U so constructed is nonempty, observe that when we remove i^* , the sum $\sum_{i \in U(t)} p(i)$ decreases by $p(i^*)$, while $\sum_{i,j \in U(t)} w(i, j)$ decreases by at most

$$\sum_{j \in U(t)} w(i^*, j) + \sum_{j \in U(t)} w(j, i^*) < rp(i^*).$$

So since $\sum_{i,j \in U(t)} w(i, j)$ was positive to begin with, it must still be positive at the end of the procedure; hence U must be nonempty. \square

We can now prove the main result of the section.

THEOREM 5.4. Suppose a snake drawn from $\mathcal{D}_{h,L}$ is ε -good with probability at least $9/10$. Then

$$\text{RLS}(G) = \Omega(1/\varepsilon), \quad \text{QLS}(G) = \Omega(\sqrt{1/\varepsilon}).$$

Proof. Given a snake $X \in \mathcal{D}_{h,L}$, we construct an input function f_X as follows. For each $v \in X$, let $f_X(v) = \min \{t : x_t = v\}$; and for each $v \notin X$, let $f_X(v) = \Delta(v, h) + L$, where $\Delta(v, h)$ is the distance from v to h in G . Clearly f_X so defined has a unique local minimum at x_0 . To obtain a decision problem, we stipulate that querying x_0 reveals an answer bit (0 or 1) in addition to $f_X(x_1)$; the algorithm's goal is then to return the answer bit. Obviously a lower bound for the decision problem implies a corresponding lower bound for the search problem.

Let us first prove the theorem in the case that all snakes in $D_{h,L}$ are ε -good. Let $p(X)$ be the probability of drawing snake X from $\mathcal{D}_{h,L}$. Also, given snakes X, Y and $j \in \{0, \dots, L-1\}$, let $q_j(X, Y)$ be the probability that $X^* = Y$, if X^* is drawn from $\mathcal{D}_{h,L}$ conditioned on agreeing with X on all steps later than j . Then define

$$w(X, Y) = \frac{p(X)}{L} \sum_{j=0}^{L-1} q_j(X, Y).$$

Our first claim is that w is symmetric; that is, $w(X, Y) = w(Y, X)$. It suffices to show that

$$p(X) q_j(X, Y) = p(Y) q_j(Y, X)$$

for all j . We can assume X agrees with Y on all steps later than j , since otherwise $q_j(X, Y) = q_j(Y, X) = 0$. Given an $X^* \in D_{h,L}$, let A denote the event that X^* agrees with X (or equivalently Y) on all steps later than j , and let B_X (resp., B_Y) denote the event that X^* agrees with X (resp., Y) on steps 1 to j . Then

$$\begin{aligned} p(X) q_j(X, Y) &= \Pr[A] \Pr[B_X|A] \cdot \Pr[B_Y|A] \\ &= p(Y) q_j(Y, X). \end{aligned}$$

Now let $E(X, Y)$ denote the event that $X \cap Y = S_{X,Y}$, where $S_{X,Y}$ is as in Definition 5.2. Also, let f_X be the input obtained from X that has answer bit 0, and g_X be the input that has answer bit 1. To apply Theorems 4.1 and 4.3, take $\mathcal{A} = \{f_X : X \in D_{h,L}\}$ and $\mathcal{B} = \{g_X : X \in D_{h,L}\}$. Then take $R(f_X, g_Y) = w(X, Y)$ if $E(X, Y)$ holds, and $R(f_X, g_Y) = 0$ otherwise. Given $f_X \in \mathcal{A}$ and $g_Y \in \mathcal{B}$ with $R(f_X, g_Y) > 0$, and letting v be a vertex such that $f_X(v) \neq g_Y(v)$, we must then have either $v \notin Y$ or $v \notin X$. Suppose the former case; then

$$\sum_{f_{X^*} \in \mathcal{A} : f_{X^*}(v) \neq g_Y(v)} R(f_{X^*}, g_Y) \leq \sum_{f_{X^*} \in \mathcal{A} : f_{X^*}(v) \neq g_Y(v)} \frac{p(Y)}{L} \sum_{j=0}^{L-1} q_j(Y, X^*) \leq \varepsilon p(Y),$$

since Y is ε -good. Thus $\theta(g_Y, v)$ equals

$$\frac{\sum_{f_{X^*} \in \mathcal{A} : f_{X^*}(v) \neq g_Y(v)} R(f_{X^*}, g_Y)}{\sum_{f_{X^*} \in \mathcal{A}} R(f_{X^*}, g_Y)} \leq \frac{\varepsilon p(Y)}{9p(Y)/10}.$$

Similarly, if $v \notin X$, then $\theta(f_X, v) \leq 10\varepsilon/9$ by symmetry. Hence

$$\begin{aligned} v_{\min} &= \max_{f_X \in \mathcal{A}, g_Y \in \mathcal{B}, v : R(f_X, g_Y) > 0, f_X(v) \neq g_Y(v)} \min\{\theta(f_X, v), \theta(g_Y, v)\} \leq \frac{\varepsilon}{9/10}, \\ v_{\text{geom}} &= \max_{f_X \in \mathcal{A}, g_Y \in \mathcal{B}, v : R(f_X, g_Y) > 0, f_X(v) \neq g_Y(v)} \sqrt{\theta(f_X, v) \theta(g_Y, v)} \leq \sqrt{\frac{\varepsilon}{9/10}}, \end{aligned}$$

the latter since $\theta(f_X, v) \leq 1$ and $\theta(g_Y, v) \leq 1$ for all f_X, g_Y , and v .

We now turn to the general case, in which a snake drawn from $\mathcal{D}_{h,L}$ is ε -good with probability at least $9/10$. Let $G(X)$ denote the event that X is ε -good. Take $\mathcal{A}^* = \{f_X \in \mathcal{A} : G(X)\}$ and $\mathcal{B}^* = \{g_Y \in \mathcal{B} : G(Y)\}$, and take $R(f_X, g_Y)$ as before. Now note that

$$\sum_{X, Y} w(X, Y) = \sum_{X, Y} p(X) p(Y) = 1,$$

so we can consider a distribution \mathcal{W} over (X, Y) pairs, in which $\Pr [(X, Y)] = w(X, Y)$. Then by the assumption of the theorem and the definition of ε -goodness,

$$\begin{aligned} \Pr_{(X,Y) \in \mathcal{W}} [G(X)] &\geq \frac{9}{10}, \\ \Pr_{(X,Y) \in \mathcal{W}} [G(Y)] &\geq \frac{9}{10}, \\ \Pr_{(X,Y) \in \mathcal{W}} [E(X, Y) \mid G(X)] &\geq \frac{9}{10}. \end{aligned}$$

Hence

$$\begin{aligned} &\sum_{f_X \in \mathcal{A}^*, g_Y \in \mathcal{B}^*} R(f_X, g_Y) \\ &= \sum_{X, Y : G(X) \wedge G(Y) \wedge E(X, Y)} w(X, Y) \\ &= \Pr_{(X,Y) \in \mathcal{W}} [G(X) \wedge G(Y) \wedge E(X, Y)] \\ &\geq 1 - \Pr_{(X,Y) \in \mathcal{W}} [\neg G(X)] - \Pr_{(X,Y) \in \mathcal{W}} [\neg G(Y)] - \Pr_{(X,Y) \in \mathcal{W}} [\neg E(X, Y) \wedge G(X)] \\ &\geq 1 - \Pr_{(X,Y) \in \mathcal{W}} [\neg G(X)] - \Pr_{(X,Y) \in \mathcal{W}} [\neg G(Y)] - \Pr_{(X,Y) \in \mathcal{W}} [\neg E(X, Y) \mid G(X)] \\ &\geq 1 - \frac{1}{10} - \frac{1}{10} - \frac{1}{10} \\ &= \frac{7}{10}, \end{aligned}$$

where the third line follows from the union bound.

So by Lemma 5.3, there exist subsets $\tilde{\mathcal{A}} \subseteq \mathcal{A}^*$ and $\tilde{\mathcal{B}} \subseteq \mathcal{B}^*$ such that for all $f_X \in \tilde{\mathcal{A}}$ and $g_Y \in \tilde{\mathcal{B}}$,

$$\begin{aligned} \sum_{g_{Y^*} \in \tilde{\mathcal{B}}} R(f_X, g_{Y^*}) &\geq \frac{7p(X)}{20}, \\ \sum_{f_{X^*} \in \tilde{\mathcal{A}}} R(f_{X^*}, g_Y) &\geq \frac{7p(Y)}{20}. \end{aligned}$$

It follows that for all f_X, g_Y with $R(f_X, g_Y) > 0$, and all v such that $f_X(v) \neq g_Y(v)$, either $\theta(f_X, v) \leq 20\varepsilon/7$ or $\theta(g_Y, v) \leq 20\varepsilon/7$. Hence $v_{\min} \leq 20\varepsilon/7$ and $v_{\text{geom}} \leq \sqrt{20\varepsilon/7}$. \square

6. Specific graphs. In this section we apply the “snake method” developed in section 5 to specific examples of graphs: the Boolean hypercube in section 6.1, and the d -dimensional cubic grid (for $d \geq 3$) in section 6.2.

6.1. Boolean hypercube. Abusing notation, we let $\{0, 1\}^n$ denote the n -dimensional Boolean hypercube—that is, the graph whose vertices are n -bit strings, with two vertices adjacent if and only if they have Hamming distance 1. Given a vertex $v \in \{0, 1\}^n$, we let $v[0], \dots, v[n-1]$ denote the n bits of v , and let $v^{(i)}$ denote the neighbor obtained by flipping bit $v[i]$. In this section we lower-bound RLS ($\{0, 1\}^n$) and QLS ($\{0, 1\}^n$).

Fix a “snake head” $h \in \{0,1\}^n$ and take $L = 2^{n/2}/100$. We define the snake distribution $\mathcal{D}_{h,L}$ via what we call a *coordinate loop*, which starts at the head $x_{L-1} = h$, and works backward to the tail x_0 as follows. For each $t \in \{L-2, \dots, 0\}$, we set $x_t := x_{t+1}$ with 1/2 probability, and $x_t := x_{t+1}^{(t \bmod n)}$ with 1/2 probability. The following is a basic fact about this distribution.

PROPOSITION 6.1. *The coordinate loop mixes completely in n steps, in the sense that if $t^* \leq t - n$, then x_{t^*} is a uniform random vertex conditioned on x_t .*

Proof. Once we reach x_{t^*} , each of the n coordinates of x_t has been replaced by a uniform random coordinate. \square

We could also use the random walk distribution, following Aldous [3]. However, not only is the coordinate loop distribution easier to work with (since it produces fewer self-intersections), it also yields a better lower bound (since it mixes completely in n steps, as opposed to approximately in $n \log n$ steps).

We first upper-bound the probability, over X, j , and $Y[j]$, that $X \cap Y \neq S_{X,Y}$ (where $S_{X,Y}$ is as in Definition 5.2).

LEMMA 6.2. *Suppose X is drawn from $\mathcal{D}_{h,L}$, j is drawn uniformly at random from $\{0, \dots, L-1\}$, and $Y[j]$ is drawn from $\mathcal{D}_{x_j,j}$. Then*

$$\Pr_{X,j,Y[j]} [X \cap Y = S_{X,Y}] \geq 0.9999.$$

Proof. We claim that for all indices $t, t^* \in \{0, \dots, L-1\}$ with $t \neq t^*$, we have

$$\Pr_{X,j,Y[j]} [x_t = y_{t^*}] \leq \frac{1}{2^n}.$$

Let us first see why the lemma follows from this claim. If there are no “collisions” of the form $x_t = y_{t^*}$ with $t \neq t^*$, then clearly

$$\min \{t : x_t = v\} = \min \{t^* : y_{t^*} = v\}$$

for all vertices v , and hence $X \cap Y = S_{X,Y}$. It follows by the union bound that

$$\begin{aligned} \Pr_{X,j,Y[j]} [X \cap Y \neq S_{X,Y}] &\leq \sum_{t \neq t^*} \Pr_{X,j,Y[j]} [x_t = y_{t^*}] \\ &\leq \sum_{t \neq t^*} \frac{1}{2^n} \\ &\leq \frac{L^2}{2^n} \\ &= 0.0001. \end{aligned}$$

It remains only to prove the claim. There are two cases. First, if $t > j - n$ and $t^* > j - n$, then $x_t \neq y_{t^*}$ for all $t \neq t^*$, and hence $\Pr_{X,j,Y[j]} [x_t = y_{t^*}] = 0$. This follows from the definition of the coordinate loop $\mathcal{D}_{h,L}$. Second, if $t \leq j - n$ or $t^* \leq j - n$, then Proposition 6.1 implies that for all vertices $v, w \in \{0,1\}^n$,

$$\Pr_{X,j,Y[j]} [x_t = v \wedge y_{t^*} = w] = \frac{1}{2^{2n}}.$$

In other words, y_{t^*} is uniformly random conditioned on x_t , and x_t is uniformly random conditioned on y_{t^*} . Therefore

$$\Pr_{X,j,Y[j]} [x_t = y_{t^*}] = \frac{1}{2^n},$$

and we are done. \square

We now argue that, unless X spends a “pathological” amount of time in one part of the hypercube, the probability of any vertex v being hit when X flicks its tail is small. To prove this, we define a notion of *sparseness*, and then show that (1) almost all snakes drawn from $\mathcal{D}_{h,L}$ are sparse (Lemma 6.4), and (2) sparse snakes are unlikely to hit any given vertex v (Lemma 6.5).

DEFINITION 6.3. *Given vertices v, w and $i \in \{0, \dots, n - 1\}$, let $\Delta(x, v, i)$ be the number of steps needed to reach v from x by first setting $x[i] := v[i]$, then setting $x[i - 1] := v[i - 1]$, and so on. (After we set $x[0]$ we wrap around to $x[n - 1]$.) Then X is sparse if there exists a constant c such that for all $v \in \{0, 1\}^n$ and all k ,*

$$|\{t : \Delta(x_t, v, t \bmod n) = k\}| \leq cn \left(n + \frac{L}{2^{n-k}} \right).$$

LEMMA 6.4. *If X is drawn from $\mathcal{D}_{h,L}$, then X is sparse with probability $1 - o(1)$.*

Proof. For each $i \in \{0, \dots, n - 1\}$, the number of $t \in \{0, \dots, L - 1\}$ such that $t \equiv i \pmod{n}$ is at most L/n . For such a t , let $E_t^{(v,i,k)}$ be the event that $\Delta(x_t, v, i) \leq k$; then $E_t^{(v,i,k)}$ holds if and only if

$$x_t[i + 1] = v[i + 1], \dots, x_t[i - k] = v[i - k]$$

(where we wrap around to $x_t[0]$ after reaching $x_t[n - 1]$). This occurs with probability $1/2^{n-k}$ over X . So let

$$\mu_k = \frac{L}{n} \cdot \frac{1}{2^{n-k}}.$$

Then for fixed v, i, k , the expected number of t 's for which $E_t^{(v,i,k)}$ holds is at most μ_k . Furthermore, by Proposition 6.1, the $E_t^{(v,i,k)}$ events for different t 's are independent. Thus by a Chernoff bound (see Motwani and Raghavan [21], for example), if $\mu_k \geq 1$, then

$$\Pr_X \left[\left| \{t : E_t^{(v,i,k)}\} \right| > cn \cdot \mu_k \right] < \left(\frac{e^{cn-1}}{(cn)^{cn}} \right)^{\mu_k} < \frac{1}{2^{2n}}$$

for sufficiently large c . Similarly, if $\mu_k < 1$, then

$$\Pr_X \left[\left| \{t : E_t^{(v,i,k)}\} \right| > cn \right] < \left(\frac{e^{cn/\mu_k-1}}{(cn/\mu_k)^{cn/\mu_k}} \right)^{\mu_k} < \frac{1}{2^{2n}}$$

for sufficiently large c . By the union bound, then,

$$\begin{aligned} \left| \{t : E_t^{(v,i,k)}\} \right| &\leq cn \cdot (1 + \mu_k) \\ &= c \left(n + \frac{L}{2^{n-k}} \right) \end{aligned}$$

for every v, i, k triple *simultaneously* with probability at least $1 - n^2 2^n / 2^{2n} = 1 - o(1)$. Summing over all i 's produces the additional factor of n . \square

LEMMA 6.5. *Let X be sparse. Suppose j is drawn uniformly from $\{0, \dots, L - 1\}$, and then $Y[j]$ is drawn from $\mathcal{D}_{x_j,j}$. Then for every $v \in \{0, 1\}^n$,*

$$\Pr_{j,Y} [v \in Y[j]] = O\left(\frac{n^2}{L}\right).$$

Proof. By assumption, for every $k \in \{0, \dots, n\}$,

$$\begin{aligned} \Pr_j [\Delta(x_j, v, j \bmod n) = k] &\leq \frac{|\{t : \Delta(x_t, v, t \bmod n) = k\}|}{L} \\ &\leq \frac{cn}{L} \left(n + \frac{L}{2^{n-k}} \right). \end{aligned}$$

Consider the probability that $v \in Y[j]$ in the event that $\Delta(x_j, v, j \bmod n) = k$. Clearly

$$\Pr_Y [v \in \{y_{j-n+1}, \dots, y_j\}] = \frac{1}{2^k}.$$

Also, Proposition 6.1 implies that for every $t \leq j - n$, the probability that $y_t = v$ is 2^{-n} . So by the union bound,

$$\Pr_Y [v \in \{y_0, \dots, y_{j-n}\}] \leq \frac{L}{2^n}.$$

Then $\Pr_{j,Y} [v \in Y[j]]$ equals

$$\sum_{k=0}^n \Pr_j [\Delta(x_j, v, j \bmod n) = k] \cdot \Pr_Y [v \in Y[j] \mid \Delta(x_j, v, j \bmod n) = k],$$

which is at most

$$\sum_{k=0}^n \frac{cn}{L} \left(n + \frac{L}{2^{n-k}} \right) \left(\frac{1}{2^k} + \frac{L}{2^n} \right) = O\left(\frac{cn^2}{L}\right),$$

as can be verified by breaking the sum into cases and doing some manipulations. \square

The main result follows easily.

THEOREM 6.6.

$$\text{RLS}(\{0, 1\}^n) = \Omega\left(\frac{2^{n/2}}{n^2}\right),$$

$$\text{QLS}(\{0, 1\}^n) = \Omega\left(\frac{2^{n/4}}{n}\right).$$

Proof. Take $\varepsilon = n^2/2^{n/2}$. Then by Theorem 5.4, it suffices to show that a snake X drawn from $\mathcal{D}_{h,L}$ is $O(\varepsilon)$ -good with probability at least $9/10$. First, since

$$\Pr_{X,j,Y[j]} [X \cap Y = S_{X,Y}] \geq 0.9999$$

by Lemma 6.2, Markov's inequality shows that

$$\Pr_X \left[\Pr_{j,Y[j]} [X \cap Y = S_{X,Y}] \geq \frac{9}{10} \right] \geq \frac{19}{20}.$$

Second, by Lemma 6.4, X is sparse with probability $1 - o(1)$, and by Lemma 6.5, if X is sparse, then

$$\Pr_{j,Y} [v \in Y[j]] = O\left(\frac{n^2}{L}\right) = O(\varepsilon)$$

for every v . So both requirements of Definition 5.2 hold simultaneously with probability at least $9/10$. \square

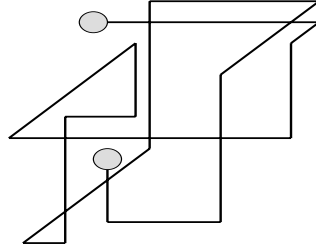


FIG. 2. In $d = 3$ dimensions, a snake drawn from $\mathcal{D}_{h,L}$ moves a random distance left or right, then a random distance up or down, then a random distance inward or outward, etc.

6.2. Constant-dimensional grid graph. In the Boolean hypercube case, we defined $\mathcal{D}_{h,L}$ by a “coordinate loop” instead of the usual random walk mainly for convenience. When we move to the d -dimensional grid, though, the drawbacks of random walks become more serious: first, the mixing time is too long, and second, there are too many self-intersections, particularly if $d \leq 4$. Our snake distribution will instead use straight lines of randomly chosen lengths attached at the endpoints, as in Figure 2. Let $G_{d,N}$ be a d -dimensional grid graph with $d \geq 3$. That is, $G_{d,N}$ has N vertices of the form $v = (v[0], \dots, v[d-1])$, where each $v[i]$ is in $\{1, \dots, N^{1/d}\}$ (we assume for simplicity that N is a d th power). Vertices v and w are adjacent if and only if $|v[i] - w[i]| = 1$ for some $i \in \{0, \dots, d-1\}$, and $v[j] = w[j]$ for all $j \neq i$ (so $G_{d,N}$ does not wrap around at the boundaries).

We take $L = \sqrt{N}/100$, and define the snake distribution $\mathcal{D}_{h,L}$ as follows. We start from the head $x_{L-1} = h$. Then for all t of the form $L - 1 - \tau N^{1/d}$, where τ is a positive integer, we set x_t identical to $x_{t+N^{1/d}}$, but with the $(\tau \bmod d)$ th coordinate $x_t[\tau \bmod d]$ replaced by a uniform random value in $\{1, \dots, N^{1/d}\}$. We then take the vertices $x_{t+N^{1/d}-1}, \dots, x_t$ to lie along the shortest path from $x_{t+N^{1/d}}$ to x_t , “stalling” at x_t once that vertex has been reached. We call

$$\Phi_\tau = (x_{t+N^{1/d}}, \dots, x_t)$$

a *line* of vertices, whose *direction* is $\tau \bmod d$. As in the Boolean hypercube case, we have the following proposition.

PROPOSITION 6.7. *$\mathcal{D}_{h,L}$ mixes completely in $dN^{1/d}$ steps, in the sense that if $t^* \leq t - d$, then $x_{t^*N^{1/d}}$ is a uniform random vertex conditioned on $x_{tN^{1/d}}$.*

Proof. Once we reach $x_{t^*N^{1/d}}$, each of the d coordinates of $x_{tN^{1/d}}$ has been replaced by a uniform random coordinate. \square

Lemma 6.2 in section 6.1 goes through essentially without change.

DEFINITION 6.8. *Let $\Delta(x, v, i)$ be the number of steps needed to reach v from x by first setting $x[i] := v[i]$, then setting $x[i+1] := v[i+1]$, and so on. (After we set $x[d-1]$ we wrap around to $x[0]$.) Also, let $\tau(t) = \lceil (L-1-t)/N^{1/d} \rceil$ be the value of τ corresponding to t . Then we say X is sparse if there exists a constant c (possibly dependent on d) such that for all vertices v and all k ,*

$$|\{t : \Delta(x_t, v, \tau(t) \bmod d) = k\}| \leq (c \log N) \left(N^{1/d} + \frac{L}{N^{1-k/d}} \right).$$

LEMMA 6.9. *If X is drawn from $\mathcal{D}_{h,L}$, then X is sparse with probability $1 - o(1)$.*

Proof. The proof is similar to Lemma 6.4. Let Φ_τ be a line of vertices with direction $i = \tau \bmod d$, and notice that $\Delta(x_t, v, i)$ is the same for every vertex x_t in Φ_τ . Let $E_\tau^{(v,i,k)}$ denote the event that $\Delta(x_t, v, i) \leq k$ for the x_t 's in Φ_τ . Then $E_\tau^{(v,i,k)}$ occurs with probability $N^{k/d}/N$ over X . Thus, let

$$\mu_k = \frac{L}{N^{1/d}} \cdot \frac{N^{k/d}}{N}.$$

Then for fixed v, i, k , the expected number of lines for which $E_\tau^{(v,i,k)}$ holds is at most μ_k . Furthermore, if $|\tau - \tau^*| \geq d$, then $E_\tau^{(v,i,k)}$ and $E_{\tau^*}^{(v,i,k)}$ are independent events. Thus, by a Chernoff bound, if $\mu_k \geq 1$, then

$$\Pr_X \left[\left| \left\{ \tau : E_\tau^{(v,i,k)} \right\} \right| > c \log N \cdot \mu_k \right] < \left(\frac{e^{c \log N - 1}}{(c \log N)^{c \log N}} \right)^{\mu_k}$$

which is at most $1/N^2$ for sufficiently large c . Similarly, if $\mu_k < 1$, then letting $m = (c \log N) / \mu_k$,

$$\Pr_X \left[\left| \left\{ \tau : E_\tau^{(v,i,k)} \right\} \right| > c \log N \right] < \left(\frac{e^{m-1}}{m^m} \right)^{\mu_k} < \frac{1}{N^2}$$

for sufficiently large c . So if we let $\tau(t) = \lceil (L - 1 - t) / N^{1/d} \rceil$ as before, then with probability $1 - o(1)$ it holds that for all v, k ,

$$\begin{aligned} |\{t : \Delta(x_t, v, \tau(t) \bmod d) = k\}| &\leq c \log N \cdot (1 + \mu_k) \cdot N^{1/d} \\ &= (c \log N) \left(N^{1/d} + \frac{L}{N^{1-k/d}} \right). \quad \square \end{aligned}$$

LEMMA 6.10. *If X is sparse, then for every $v \in G_{d,N}$,*

$$\Pr_{j,Y} [v \in Y[j]] = O\left(\frac{N^{1/d} \log N}{L}\right),$$

where the big O hides a constant dependent on d .

Proof. As in Lemma 6.5, setting $\tau(j) = \lceil (L - 1 - j) / N^{1/d} \rceil$ we obtain that

$$\begin{aligned} \Pr_{j,Y} [v \in Y[j]] &= \sum_{k=1}^d \Pr_j [\Delta(x_j, v, \tau(j) \bmod d) = k] \Pr_Y [v \in Y[j] \mid \Delta(x_j, v, \tau(j) \bmod d) = k] \\ &\leq \sum_{k=1}^d \frac{c \log N}{L} \left(N^{1/d} + \frac{L}{N^{1-k/d}} \right) \left(\frac{1}{N^{(k-1)/d}} + \frac{L}{N} \right) \\ &= O\left(\frac{N^{1/d} \log N}{L}\right). \quad \square \end{aligned}$$

Taking $\varepsilon = (\log N) / N^{1/2-1/d}$ we get, by the same proof as for Theorem 6.6, the following theorem.

THEOREM 6.11. *Neglecting a constant dependent on d , for all $d \geq 3$,*

$$\text{RLS}(G_{d,N}) = \Omega\left(\frac{N^{1/2-1/d}}{\log N}\right),$$

$$\text{QLS}(G_{d,N}) = \Omega\left(\sqrt{\frac{N^{1/2-1/d}}{\log N}}\right).$$

Acknowledgments. I thank Andris Ambainis for suggesting an improvement to Theorem 3.2; David Aldous, Christos Papadimitriou, Yuval Peres, and Umesh Vazirani for discussions during the early stages of this work; and Ronald de Wolf and the anonymous reviewers for helpful comments.

REFERENCES

- [1] S. AARONSON, *Quantum lower bound for the collision problem*, in Proceedings of ACM STOC, 2002, pp. 635–642 (quant-ph/0111102).
- [2] D. AHARONOV AND O. REGEV, *Lattice problems in NP intersect coNP*, in Proceedings of IEEE FOCS, 2004, pp. 362–371.
- [3] D. ALDOUS, *Minimization algorithms and random walk on the d -cube*, Ann. Probab., 11 (1983), pp. 403–413.
- [4] A. AMBAINIS, *Quantum lower bounds by quantum arguments*, J. Comput. System Sci., 64 (2002), pp. 750–767 (earlier version in ACM STOC 2000, quant-ph/0002066).
- [5] A. AMBAINIS, *Polynomial degree vs. quantum query complexity*, in Proceedings of IEEE FOCS, 2003, pp. 230–239 (quant-ph/0305028).
- [6] T. BAKER, J. GILL, AND R. SOLOVAY, *Relativizations of the $P=?NP$ question*, SIAM J. Comput., 4 (1975), pp. 431–442.
- [7] R. BEALS, H. BUHRMAN, R. CLEVE, M. MOSCA, AND R. DE WOLF, *Quantum lower bounds by polynomials*, J. ACM, 48 (2001), pp. 778–797 (earlier version in IEEE FOCS 1998, pp. 352–361, quant-ph/9802049).
- [8] C. H. BENNETT, E. BERNSTEIN, G. BRASSARD, AND U. VAZIRANI, *Strengths and weaknesses of quantum computing*, SIAM J. Comput., 26 (1997), pp. 1510–1523 (quant-ph/9701001).
- [9] H. BUHRMAN AND R. DE WOLF, *Complexity measures and decision tree complexity: A survey*, Theoret. Comput. Sci., 288 (2002), pp. 21–43.
- [10] R. DIESTEL, *Graph Theory*, 2nd ed., Springer-Verlag, New York, 2000.
- [11] S. DROSTE, T. JANSEN, AND I. WEGENER, *Upper and lower bounds for randomized search heuristics in black-box optimization*, Theory Comput. Syst., to appear.
- [12] C. DÜRR AND P. HØYER, *A quantum algorithm for finding the minimum*, 1996 (quant-ph/9607014).
- [13] E. FARHI, J. GOLDSTONE, S. GUTMANN, J. LAPAN, A. LUNDGREN, AND D. PREDA, *A quantum adiabatic evolution algorithm applied to random instances of an NP-complete problem*, Science, 292 (2001), pp. 472–476 (quant-ph/0104129).
- [14] D. S. JOHNSON, C. H. PAPADIMITRIOU, AND M. YANNAKAKIS, *How easy is local search?*, J. Comput. System Sci., 37 (1988), pp. 79–100.
- [15] I. KERENIDIS AND R. DE WOLF, *Exponential lower bound for 2-query locally decodable codes via a quantum argument*, in Proceedings of ACM STOC, 2003, pp. 106–115 (quant-ph/0208062).
- [16] S. LAPLANTE AND F. MAGNIEZ, *Lower bounds for randomized and quantum query complexity using Kolmogorov arguments*, in Proceedings of IEEE Conference on Computational Complexity, 2004, pp. 294–304 (quant-ph/0311189).
- [17] D. C. LLEWELLYN AND C. TOVEY, *Dividing and conquering the square*, Discrete Appl. Math., 43 (1993), pp. 131–153.
- [18] D. C. LLEWELLYN, C. TOVEY, AND M. TRICK, *Local optimization on graphs*, Discrete Appl. Math., 23 (1989), pp. 157–178.
- [19] D. C. LLEWELLYN, C. TOVEY, AND M. TRICK, *Erratum: Local optimization on graphs*, Discrete Appl. Math., 46 (1993), pp. 93–94.
- [20] N. MEGIDDO AND C. H. PAPADIMITRIOU, *On total functions, existence theorems, and computational complexity*, Theoret. Comput. Sci., 81 (1991), pp. 317–324.

- [21] R. MOTWANI AND P. RAGHAVAN, *Randomized Algorithms*, Cambridge University Press, Cambridge, UK, 1995.
- [22] C. H. PAPADIMITRIOU, talk at UC Berkeley, February 6, 2003.
- [23] M. SANTHA AND M. SZEGEDY, *Quantum and classical query complexities of local search are polynomially related*, in Proceedings of ACM STOC, 2004, pp. 494–501.
- [24] Y. SHI, *Quantum lower bounds for the collision and the element distinctness problems*, in Proceedings of IEEE FOCS, 2002, pp. 513–519. quant-ph/0112086.
- [25] Y. VERHOEVEN, *Enhanced algorithms for local search*, Inform. Process. Lett., to appear (quant-ph/0506019).
- [26] R. ŠPALEK AND M. SZEGEDY, *All quantum adversary methods are equivalent*, in Proceedings of the International Colloquium on Automata, Languages, and Programming (ICALP), 2005, pp. 1299–1311 (quant-ph/0409116).
- [27] S. ZHANG, *(Almost) tight bounds for randomized and quantum local search on hypercubes and grids*, 2005 (quant-ph/0504085).

APPROXIMATING FRACTIONAL PACKINGS AND COVERINGS IN $O(1/\epsilon)$ ITERATIONS*

D. BIENSTOCK[†] AND G. IYENGAR[†]

Abstract. We adapt a method proposed by Nesterov [*Math. Program. Ser. A*, 103 (2005), pp. 127–152] to design an algorithm that computes ϵ -optimal solutions to fractional packing problems by solving $O(\epsilon^{-1}\sqrt{Kn\ln(m)})$ separable convex quadratic programs, where n is the number of variables, m is the number of constraints, and K is the maximum number of nonzero elements in any constraint. We show that the quadratic program can be approximated to any degree of accuracy by an appropriately defined piecewise-linear program. For the special case of the maximum concurrent flow problem on a graph $G = (V, E)$ with rational capacities and demands, we obtain an algorithm that computes an ϵ -optimal flow by solving shortest path problems, i.e., problems in which the number of shortest paths computed grows as $O(\epsilon^{-1}\log(\epsilon^{-1}))$ in ϵ and polynomially in the size of the problem. In contrast, previous algorithms required $\Omega(\epsilon^{-2})$ iterations. We also describe extensions to the maximum multicommodity flow problem, the pure covering problem, and mixed packing-covering problem.

Key words. approximation algorithms, packing problems, multicommodity flows

AMS subject classifications. 68W25, 68W40, 90C59, 90C47, 90C05, 90C06

DOI. 10.1137/S0097539705447293

1. Packing problems. The prototypical example of the problems considered in this paper is the *maximum concurrent flow* problem defined as follows. Given a graph $G = (V, E)$, where each edge $e \in E$ has a positive capacity u_e , and K commodities with associated demands $d_k \in \mathbf{R}^{|V|}$, $k = 1, \dots, K$, solve

$$\begin{aligned} \min \quad & \max_{e \in E} \left\{ \frac{\sum_{k=1}^K f_{k,e}}{u_e} \right\} \\ \text{s.t.} \quad & Nf_k = d_k, \quad k = 1, \dots, K, \\ & f_k \geq 0, \quad k = 1, \dots, K, \end{aligned}$$

where $N \in \mathbf{R}^{|V| \times |E|}$ is the node-arc incidence of the graph. Here and below we will assume that commodities are grouped by source; i.e., for each k there is one vertex $s(k)$ with $d_{k,s(k)} > 0$. We will call a nonnegative vector $f_k \in \mathbf{R}^{|E|}$ a *flow* vector for commodity k if $Nf_k = d_k$, i.e., it routes the corresponding demand d_k . A nonnegative vector $f = (f_1, f_2, \dots, f_K) \in \mathbf{R}^{K|E|}$ is called a *flow* if f_k is a flow vector for commodity k , $k = 1, \dots, K$. For a given flow f , the quantity $(\sum_k f_{k,e})/u_e$ is called the *load* or *congestion* on the edge $e \in E$. Thus, the maximum concurrent flow problem computes a flow that minimizes the maximum congestion. Problems of this nature arise in the context of routing in capacitated networks. Furthermore, many network design algorithms solve maximum concurrent flow problems as subroutines in order to find a feasible routing.

*Received by the editors September 20, 2004; accepted for publication (in revised form) September 2, 2005; published electronically February 21, 2006. A preliminary version of this paper appeared in *Proceedings of the 36th Annual Symposium on Theory of Computing*, ACM, New York, pp. 146–155. <http://www.siam.org/journals/sicomp/35-4/44729.html>

[†]IEOR Department, Columbia University, New York, NY 10027 (dano@columbia.edu, garud@ieor.columbia.edu). The research of the first author was partially supported by NSF awards CCR-ITR-0213844 and DMI-0200221. The research of the second author was partially supported by NSF grants CCR-00-09972 and DMS-01-04282, and by ONR grant N000140310514.

The maximum concurrent flow problem is a special case of the *fractional packing* problem. Let $A \in [a_1, \dots, a_m]^T \in \{0, 1\}^{m \times n}$, and let $Q \subseteq \mathbf{R}_+^n$ be a polyhedron. For $x \in Q$, let $\lambda(x) \triangleq \max_{1 \leq i \leq m} \{a_i^T x\}$. Then the fractional packing problem $\text{PACK}(A, Q)$ is given by

$$(1) \quad \lambda_{A,Q}^* \triangleq \min_{x \in Q} \lambda(x).$$

The fractional packing problem is itself a special case of the *generalized packing* problem, where each entry a_{ij} of the matrix A is assumed to be nonnegative, rather than $a_{ij} \in \{0, 1\}$.

A special case of the generalized packing problem of particular interest is the so-called *block-angular* problem. In such problems, there exist positive integers $k > 0$, and $n_i, i = 1, \dots, k$, with $\sum_i n_i = n$, such that the set $Q = Q^1 \times Q^2 \times \dots \times Q^k$, $Q^i \subseteq \mathbf{R}^{n_i}, i = 1, \dots, k$. When Q is a polyhedron this simply says that the nonzeros in the constraint matrix defining Q are arranged, without loss of generality, into a block-diagonal structure with k blocks. The maximum concurrent flow problem is clearly a block-angular problem with each block corresponding to a commodity; the constraints in a given block describe flow conservation and nonnegativity of the flow variables for the corresponding commodity.

Generalized packing problems are linear programs, and therefore can be solved to optimality in polynomial time. Nevertheless, these problems—and in particular the maximum concurrent flow problem—have long been recognized as extremely challenging. State-of-the-art implementations of the simplex method and also of interior point methods, running on fast machines, can require an inordinate amount of time to solve maximum concurrent flow problems on networks with a few thousand nodes, even when physical memory is adequate. Frequently, these codes can also consume an extremely large amount of memory. Consequently, they often prove unusable. For more background see [3].

It is not clear, from a theoretical standpoint, why traditional linear programming methods tend to require a large number of iterations to solve maximum concurrent flow problems. However, it is fairly clear why such methods tend to require a large running *time*. This is due to very expensive matrix computations, e.g., matrix inversions or Cholesky factorizations, that must be repeatedly carried out. As an example, consider a maximum concurrent flow problem on a network with $|V|$ vertices, $|E|$ edges, and K commodities. The overall constraint matrix for the linear program will have $|E| + K|V|$ rows and $K|E|$ variables. Even when the graph is very sparse ($|E| = \theta(|V|)$), if we have $K = \theta(|V|)$ the numerical linear algebra will be over matrices of dimension $\theta(|V|^2)$. Further, even though the constraint matrix may be very sparse, the matrix computations will usually experience “fill-in.”

These facts were recognized early on, in the context of block-angular linear programs, and were a motivating factor for some of the earliest decomposition ideas in mathematical programming, such as the Dantzig–Wolfe decomposition and Lagrangian relaxation. See [15, 3] for further background. The goal of these methods was to try to reduce the solution of the linear program $\min\{c^T x : Ax \leq b, x \in Q\}$ to a set of linear programs over Q . For a block-angular problem, a linear program over Q reduces to a set of linear programs over the blocks; therefore, when there are many blocks and each block is small relative to the overall Q , each iteration becomes cheap, while direct solution of the complete problem using a standard method may be impossible. For the concurrent flow problem, a linear program over Q reduces to

a set of shortest path problems for each commodity. In essence, this approach leads to a model of computation in which optimization over Q is viewed as the building block of more complex algorithms; the critical theoretical issue then becomes that of minimizing the number of iterations needed to achieve convergence.

This model of computation is perhaps not as precise as one would like it to be—we need to clearly state what can be done in addition to optimization over Q . Nevertheless, the model does capture the notion that we want to avoid matrix algebra over large matrices. In this paper we will adopt this model (an algorithm that fits the model will be called a *decomposition method*), and achieving low iteration counts will be a central goal in the design of our algorithms.

1.1. Modern results. The difficulty of the maximum concurrent flow problem, and its practical relevance, have long provided strong motivation for developing fast algorithms that can provide provably good, if not optimal, solutions.

Shahrokhi and Matula [26] developed the first approximation algorithm for the maximum concurrent flow problem. They considered the special case in which the capacity on all edges are equal. The method in [26] considers an exponential potential function of the form

$$\sum_{e \in E} e^{\alpha(\sum_k f_{k,e})}.$$

We will call a flow f with maximum load at most $(1 + \epsilon)$ times the optimal maximum load an ϵ -optimal flow. It is shown in [26] that, given $\epsilon \in (0, 1)$, one can choose $\alpha = \alpha(\epsilon)$ such that ϵ -optimal flow can be computed by minimizing the potential function to within an appropriate *absolute* error. The algorithm employed in [26] is, roughly, a first-order procedure to minimize the potential function, i.e., a procedure that approximates the potential function by its gradient, and the number of iterations required is $O(\epsilon^{-3})$ times a polynomial in the number of nodes and edges. In addition, the algorithm maintains a list of $O(\epsilon^{-2})$ paths so that the dependence of the running time on ϵ is $O(\epsilon^{-5})$. Each iteration consists of a shortest path computation or a single-commodity minimum-cost flow problem; thus, this algorithm is a decomposition method.

The Shahrokhi–Matula result spurred a great deal of research that generalized the techniques to broader packing and covering problems, gradually reducing the dependence of the iteration count on ϵ to finally obtain ϵ^{-2} , reducing the overall complexity to $O(\epsilon^{-2})$ times a polynomial in the size of the graph, and also simplifying the overall approach. See [15, 19, 9, 10, 24, 25, 8, 5] for details; [25, 8, 5] attain the best bounds for the maximum concurrent flow problem. All of these algorithms rely, sometimes implicitly, on the exponential potential function, and can be viewed as first-order methods. Villavicencio and Grigoriadis [30] use a logarithmic potential function. Bienstock and Raskin [4] show that the “flow deviation” algorithm for the maximum concurrent flow problem in [7] yields an $O(\epsilon^{-2})$ algorithm—this time using a rational barrier function.

A natural issue is that of proving lower bounds on the number of iterations needed to obtain an ϵ -optimal solution to a packing problem, and, in particular, the dependence of the iteration count on ϵ . Klein and Young [16] studied the complexity of $\text{PACK}(A, Q)$, assuming that there exists an oracle that, given $c \in \mathbf{R}^n$, returns an optimal *extreme point* for the linear program $\min\{c^T x : x \in Q\}$. The main result in [16] is that (under appropriate assumptions) the number of oracle calls needed to find an ϵ -optimal solution to a packing problem can be as large as $\Omega(\rho\epsilon^{-2} \log m)$. Here

$\rho = \max_{x \in Q} \max_{1 \leq i \leq m} a_i^T x$ (known as the *width* of the problem). This result applies when $\rho \epsilon^{-2} = O(m^{0.5-\delta})$ for some $\delta > 0$. Thus, in essence, the result amounts to an $\Omega(m^{0.5})$ lower bound on the iteration count. The dependence on ϵ implied by the bound is striking, and it raises the question of whether ϵ^{-2} is indeed a lower bound on the dependence on ϵ for the number of iterations required by a decomposition method. We note here that the assumption that the optimization oracle returns an *extreme point* of Q is critical in the proof in [16].

For some time, a parallel line of research has been followed in the nondifferentiable optimization community. Roughly stated, the focus of the work has been to produce solutions with small *absolute* error, rather than a small *relative* error as has been the focus of the algorithms community. This line of work has produced a number of interesting ideas and results (see [3] for references). Among them are decomposition algorithms that solve the generalized packing problem within an additive error of ϵ in $O(\epsilon^{-2})$ iterations; however, these are not polynomial-time algorithms, even for fixed ϵ , in that the $O()$ notation hides constants that depend on the matrix A and the set Q . Recently Nesterov [22] (also see [21]) obtained a major result: a decomposition algorithm that solves min-max problems to within an additive error of ϵ in $O(\epsilon^{-1})$ iterations. Again, this algorithm is, in general, not a polynomial-time algorithm, even for fixed ϵ . The algorithm combines, in a novel way, the (essentially folkloric) idea of making use of old iterate information with that of an approximate second-order approximation of the exponential potential function. In section 2.2 we present a streamlined analysis of the key ideas in [22].

1.2. New results. We show how to adapt the technique in [22] to obtain an ϵ -optimal solution (i.e., a solution with a *relative* error ϵ) to $\text{PACK}(A, Q)$ by solving at most

$$O\left(\epsilon^{-1} \sqrt{Kn \log m}\right)$$

convex, separable quadratic programs (QPs) over sets of the form

$$(2) \quad Q(\lambda_u) \triangleq \{x \in Q : 0 \leq x_j \leq \lambda_u, 1 \leq j \leq n\},$$

where $\lambda_u > 0$, and K denotes the maximum number of nonzero terms in any row of A . While some of the key ideas in this paper are derived from those in [22], our paper is self-contained. We also adapt a binary search technique from [9]. It is quite possible that in our iteration bound some of the constants and the dependency on m , n , and K can be improved with a somewhat more complex analysis. Also note that in the block-angular case each optimization over $Q(\lambda)$ breaks up into a separate problem over each block.

It is known that convex QPs can be solved in polynomial time (for some of the earliest references, see [17, 18]). But there is more that can be said here. Let S be a closed convex body in \mathbf{R}^n . Given $c \in \mathbf{R}^n$, the *linear optimization problem* over S consists of finding $\hat{x} \in S$ such that $c^T \hat{x} = \min \{c^T x : x \in S\}$. Also, given $y \in \mathbf{R}^n$, the *separation problem* over S consists of either showing that $y \in S$ or finding a violated inequality, e.g., a vector $d \in \mathbf{R}^n$ and a scalar d_0 such that $d^T y < d_0$ but $d^T x \geq d_0$ for all $x \in S$. It is well known (see [12, 13]) that the linear optimization problem and the separation problem over S are polynomially equivalent under appropriate (technical) assumptions regarding S ; i.e., any instance of the linear optimization (resp., separation) problem can be reduced to a polynomial number of instances of the separation (resp., linear optimization) problem.

Returning to our problem, suppose that Q is given by a (linear) optimization oracle, i.e., a subroutine that solves the linear optimization problem over Q for any objective vector c . For example, if Q describes the set of feasible flows in a network that correspond to sending one unit of flow from a certain vertex s to another vertex t , the subroutine would consist of an s - t shortest path algorithm. It follows that the separation problem over $Q(\lambda)$ can be solved by performing at most a polynomial number of oracle calls: the conditions $0 \leq x_j \leq \lambda_u$ can be checked directly, producing a violated inequality when not satisfied, and the condition $x \in Q$ is handled by using the aforementioned polynomial equivalence between linear optimization and separation. Under appropriate assumptions [27] (see also [17], which was also published as [18]) this implies that a convex QP over $Q(\lambda)$ can be solved by making a polynomial number of calls to the (linear) optimization oracle for Q .

The generalized packing problem can be reduced to a fractional packing problem as follows. Let N be the number of nonzeros in A . For each entry $a_{ij} > 0$, introduce a new variable y_{ij} and a new constraint

$$(3) \quad y_{ij} - a_{ij}x_{ij} = 0.$$

Let

$$P = \{y \in \mathbf{R}^N : \exists x \in Q \text{ such that (3) holds } \forall a_{ij} > 0\}.$$

It follows that

$$(4) \quad \lambda_{A,Q}^* = \min_{y \in P} \max_{1 \leq i \leq m} \sum_{j : a_{ij} > 0} y_{ij}.$$

Furthermore, P is closed convex, and because of (3), a convex separable QP over P reduces to one over Q . Hence, our result for fractional packing problems implies that, for any $\epsilon \in (0, 1)$, the number of iterations required to compute an ϵ -optimal solution for the generalized packing problem is

$$O\left(\epsilon^{-1} \sqrt{KN \log m}\right).$$

Using ideas derived from [20], we show how the QPs can be approximated by piecewise-linear programs. For the special case of maximum concurrent flows, this leads to an algorithm that computes ϵ -optimal flow by solving

$$O^* \left(\epsilon^{-1} K^2 E^{\frac{1}{2}} \left(L_U + |E| \lceil \log D \rceil + |E| \log \left(\frac{1}{\epsilon} \right) \right) + \epsilon^{-1} |V| |E| \right)$$

shortest path problems, where L_U denotes the number of bits needed to store the capacities, D is the sum of all demands, and K is the number of commodities grouped by source.

This paper is organized as follows. Section 2 presents our results on packing problems. Section 3 specializes our results to the maximum concurrent flow problem with rational capacities and demands. Section 4 describes the application of our techniques to the maximum multicommodity flow algorithms. Section 5 considers covering problems, and section 6 considers mixed packing and covering problems.

1.3. Comparison with previous results. First, we note that our results appear to violate the Klein–Young lower bound of $\Omega(\epsilon^{-2})$. This is not a contradiction since our algorithms bypass the requirements needed for the analysis in [16] to hold; in particular, we dynamically change the bounds on the variables. This feature will, in general, result in iterates in the *interior* of the set Q . In addition, some of our algorithms solve QPs over Q , again leading to interior solutions.

The previous fastest algorithms for finding ϵ -optimal solutions to generalized packing problems have a worst-case complexity that grows in proportion to ϵ^{-2} . Our methods improve on this, but at the cost of increasing the dependence on n and m . Hence, for ϵ fixed or moderately small, our algorithms are *not* the fastest. Note that, for example, the algorithm for maximum concurrent flows in [5] has complexity $O^*(\epsilon^{-2}|E|(K + |E|))$, where K is the number of origin-destination commodity pairs. See also [25, 8] for similar bounds.

Another class of algorithms worthy of comparison consists of the standard interior point polynomial-time algorithms for linear programming. Even though these are certainly not decomposition methods, their respective running times have a dependence on ϵ that grows like $\log(\epsilon^{-1})$, and hence for very small ϵ these algorithms are certainly faster than ours. Within this class, the algorithms proposed by Kapoor and Vaidya [14] and Vaidya [28, 29] deserve special attention. These algorithms are *not* decomposition algorithms, and when applied to a packing problem, Vaidya [29] computes, as a subroutine, the inverse of $n \times n$ matrices ([15] restates this as $m \times m$). In spite of this, it is possible that the algorithms in [14, 28, 29] are faster than ours even for moderate ϵ .

In summary, the primary contribution of this paper is to show that there exist decomposition methods for which the iteration count for computing ϵ -optimal solutions to generalized packing problems (among others) grows like $O(\epsilon^{-1})$ and is polynomial in n and m . We expect that the actual running *time* of our algorithms can be improved by using ideas such as those in [25, 8, 5] and, possibly, by using randomization as in [15].

1.4. Notation. In this paper, an ϵ -optimal solution will denote a solution that has a *relative* error at most ϵ . We also compute solutions with a given *absolute* error. We use the $O(\cdot)$ notation to denote the leading term in ϵ and the $O^*(\cdot)$ notation to suppress polylog factors in parameters such as m , n , and K . Also, since our focus in this paper is on designing decomposition algorithms with improved dependence on ϵ , we will occasionally state $O(\cdot)$ and $O^*(\cdot)$ bounds in terms of their dependence on ϵ , while suppressing polynomial factors in m , n , and K .

2. Fractional packing problems $\text{PACK}(A, Q)$. In this section we present our algorithm for $\text{PACK}(A, Q)$. As with many of the previous algorithms, our algorithm consists of an outer binary search procedure for refining an estimate for $\lambda_{A,Q}^*$ and an inner procedure that is a potential reduction algorithm.

2.1. Outer loop: Binary search. In this section we abbreviate $\lambda_{A,Q}^*$ as λ^* . We assume that $\lambda^* > 0$ and that we are given bounds $0 < \lambda_l \leq \lambda^* \leq \lambda_u$ such that $\lambda_u \leq O(\min\{m, K\})\lambda_l$. Such bounds can be computed by solving a polynomial (and independent of ϵ) number of linear programs over sets $Q(\lambda)$; see [3, 9] for details. Next, these bounds are refined using a binary search procedure introduced in [9] (see [3] for an alternate derivation). In the binary search procedure below, $\text{ABSOLUTE}(Q, A, \lambda_u, \delta)$ denotes any algorithm that returns an $x \in Q$ such that $\lambda(x) \leq \lambda^* + \delta$, i.e., an x that has an *absolute* error less than δ .

BINARY SEARCH

```

Input: values  $(\lambda_l, \lambda_u)$  with  $\lambda_l \leq \lambda^* \leq \lambda_u \leq 2 \min\{m, K\}\lambda_l$ 
Output:  $\hat{y} \in Q$  such that  $\lambda(y) \leq (1 + \epsilon)\lambda^*$ 
while  $(\lambda_u - \lambda_l) \geq \epsilon\lambda_l$  do
    set  $\delta = \frac{1}{3}(\lambda_u - \lambda_l)$ 
    set  $\hat{x} \leftarrow \text{ABSOLUTE}(Q, A, \lambda_u, \delta)$ 
    if  $\lambda(\hat{x}) \geq \frac{1}{3}\lambda_l + \frac{2}{3}\lambda_u$ , set  $\lambda_l \leftarrow \frac{2}{3}\lambda_l + \frac{1}{3}\lambda_u$ 
    else set  $\lambda_u \leftarrow \frac{1}{3}\lambda_l + \frac{2}{3}\lambda_u$ 
return  $\hat{x}$ 
    
```

In section 2.2 we construct an algorithm ABSOLUTE with the following properties.

THEOREM 1. *There exists an algorithm $\text{ABSOLUTE}(A, Q, \lambda_u, \delta)$ that computes, for any $\delta \in (0, \lambda_u)$, an $\hat{x} \in Q$ with $\lambda(\hat{x}) \leq \lambda^* + \delta$ by solving $O(\sqrt{Kn \log m} \frac{\lambda_u}{\delta})$ separable convex QPs over $Q(\lambda_u)$.*

As a consequence, we have the following complexity bound for the above binary search procedure.

COROLLARY 2. *The complexity of the binary search procedure is*

$$O\left(\epsilon^{-1} C_q \sqrt{Kn \log m}\right),$$

where C_q is the cost of solving a convex separable QP over $Q(\lambda_u)$.

Proof of Corollary 2. Consider first the iterations, where $\lambda_u > 2\lambda_l$. Since initially

$$\lambda_u \leq O(\min\{m, K\})\lambda_l,$$

there are at most $O(\log \min\{m, K\})$ iterations of this sort. In each such iteration we have $\frac{\lambda_u}{\delta} < 6$; thus by Theorem 1 the total number of QPs over sets $Q(\lambda_u)$ that are solved during these iterations is polynomial in K, n , and m .

Now consider the remaining iterations, i.e., those where $\lambda_u \leq 2\lambda_l$. Denoting by H the number of such iterations, we have $H = O(\log(1/\epsilon))$. Further, it is easy to check that in each iteration the ratio λ_u/δ increases by at most a factor of $3/2$. Now, again by Theorem 1, the number of QPs solved during the remaining iterations is, up to constants,

$$\sqrt{Kn \log m} \sum_{h=0}^H \left(\frac{3}{2}\right)^h = 2\sqrt{Kn \log m} \left(\left(\frac{3}{2}\right)^H - 1\right).$$

Thus, the result follows. \square

2.2. Inner loop: ABSOLUTE(A, Q, λ_u, δ). In this section we describe the ABSOLUTE(A, Q, λ_u, δ) algorithm. We construct this algorithm using techniques from [22].

Define $\bar{Q} \in \mathbf{R}^{2n}$ as

$$\bar{Q} \triangleq \{(x, y) \in \mathbf{R}^{2n} : x \in Q, x_j = \lambda_u y_j, y_j \leq 1, j = 1, \dots, n\}.$$

Since $\lambda^* \leq \lambda_u$, $\bar{Q} \neq \emptyset$. Let P denote the projection of \bar{Q} onto the space of the y variables, i.e.,

$$P = \{y \in \mathbf{R}^n : \exists x \text{ s.t. } (x, y) \in \bar{Q}\}.$$

If the feasible set Q in (1) is a polyhedron, so are \bar{Q} and P . Moreover, $P \subseteq [0, 1]^n$. Define

$$\lambda_P^* \triangleq \min \left\{ \lambda(y) = \max_{1 \leq i \leq m} \{a_i^T y\} : y \in P \right\}.$$

Then it follows that $\lambda_P^* = \lambda^*/\lambda_u \leq 1$.

In this section we describe an algorithm that, for any $\gamma \in (0, 1)$, computes $\hat{y} \in P$, with $\lambda(\hat{y}) \leq \lambda_P^* + \gamma$, by solving $O(\gamma^{-1}\sqrt{Kn \log m})$ separable convex QPs over P . Note that these programs reduce to separable convex QPs over $Q(\lambda_u)$, and, in the case of multicommodity flow problems, break up into separable convex quadratic min-cost flow problems over each commodity. Choosing $\gamma = \frac{\delta}{\lambda_u}$ will accomplish the objective of this section.

2.2.1. Potential reduction algorithm. For the purposes of this section, we assume that we are given a matrix $A \in \{0, 1\}^{m \times n}$ with at most K nonzeros per row, a (nonempty) closed convex set $P \subseteq [0, 1]^n$, and a constant $\gamma \in (0, 1)$. Furthermore, it is known that $\lambda_P^* \leq 1$.

Define the *potential function* $\Phi(x)$ as follows [9, 24]:

$$\Phi(x) \triangleq \frac{1}{|\alpha|} \ln \left(\sum_i e^{\alpha a_i^T x} \right), \quad \alpha \neq 0.$$

It is easy to show (see [9] for details) that for $\alpha > 0$,

$$(5) \quad \lambda(x) \leq \Phi(x) \leq \lambda(x) + \frac{\ln m}{\alpha} \quad \forall x \in P.$$

Let $\Phi^* \triangleq \min\{\Phi(x) : x \in P\}$ and choose $\alpha = \frac{2 \ln m}{\gamma}$. Then, for all $x \in P$ such that $\phi(x) \leq \Phi^* + \gamma/2$, the bound (5) implies that

$$\lambda(x) \leq \Phi(x) \leq \Phi^* + \frac{\gamma}{2} \leq \Phi(x^*) + \frac{\gamma}{2} \leq \lambda^* + \frac{\ln m}{\alpha} + \frac{\gamma}{2} = \lambda^* + \gamma.$$

Thus, our task reduces to computing $x \in P$ satisfying $\Phi(x) \leq \Phi^* + \gamma/2$.

In what follows, the notation $\langle x, y \rangle$ will denote the usual Euclidean inner product $x^T y$.

ALGORITHM QP

Input: $P \subseteq [0, 1]^n$, $\frac{0}{1}$ matrix A , $\alpha = \frac{\ln m}{\gamma}$, $L = \gamma^{-1}\sqrt{8Kn \ln(m)}$
Output: $\hat{y} \in P$ such that $\lambda(\hat{y}) \leq \lambda_P^* + \gamma$
choose $x^{(0)} \in P$. **set** $t \leftarrow 0$.
while ($t \leq L$) **do**

$g^{(t)} \leftarrow \nabla \Phi(x^{(t)})$

$y^{(t)} \leftarrow \operatorname{argmin}_{x \in P} \left\{ \frac{K|\alpha|}{2} \sum_{j=1}^n (x_j - x_j^{(t)})^2 + \langle g^{(t)}, x - x^{(t)} \rangle \right\}$

$S_t(x) \leftarrow \frac{2}{(t+1)(t+2)} \left\{ K|\alpha| \sum_{j=1}^n (x_j - x_j^0)^2 + \sum_{h=0}^t (h+1) [\Phi(x^{(h)}) + \langle g^{(h)}, x - x^{(h)} \rangle] \right\}$

$z^{(t)} \leftarrow \operatorname{argmin}_{x \in P} \{S_t(x)\}$

$x^{(t+1)} \leftarrow \left(\frac{2}{t+3} \right) z^{(t)} + \left(\frac{t+1}{t+3} \right) y^{(t)}$

$t \leftarrow t + 1$

return $y^{(L)}$

The following result is established in section 2.2.2.

THEOREM 3. For any $t \geq 0$, $\Phi(y^{(t)}) \leq S_t(z^{(t)})$.

Theorem 3 implies the following corollary.

COROLLARY 4. $\Phi(y^{(L)}) \leq \Phi^* + \gamma/2$.

Proof. Fix $t \geq 0$. Let $x^* = \operatorname{argmin}_{x \in P} \{\Phi(x)\}$. By definition,

$$\begin{aligned} S_t(z^t) &\leq S_t(x^*), \\ &= \frac{2}{(t+1)(t+2)} \left\{ K\alpha \sum_{j=1}^n (x_j^* - x_j^{(0)})^2 + \sum_{h=0}^t (h+1) [\Phi(x^{(h)}) + \langle g^{(h)}, x^* - x^{(h)} \rangle] \right\} \\ (6) \quad &\leq \frac{2Kn\alpha}{(t+1)(t+2)} + \Phi^*, \end{aligned}$$

where (6) follows from the following facts:

$$\begin{aligned} 0 \leq x_i^*, x_i^{(0)} \leq 1 &\Rightarrow (x_i^* - x_i^{(0)}) \leq 1, \\ \Phi \text{ convex} &\Rightarrow \Phi(x^{(h)}) + \langle g^{(h)}, x^* - x^{(h)} \rangle \leq \Phi(x^*). \end{aligned}$$

Theorem 3 implies that $\Phi(y^{(t)}) \leq S_t(z^{(t)})$. Thus,

$$\Phi(y^{(t)}) - \Phi^* \leq \frac{\gamma}{2m}, \quad t \geq L = \frac{\sqrt{8Kn \ln m}}{\gamma}. \quad \square$$

2.2.2. Proof of Theorem 3. The following result is a refinement of the Taylor series bound in [9] (see also [3, 23]). The proof is provided in section 7.1.

LEMMA 5. For all $x, y \in P$, we have $\Phi(y) \leq \Phi(x) + \langle \nabla \Phi(x), y - x \rangle + \frac{K|\alpha|}{2} \sum_j (y_j - x_j)^2$.

The rest of the proof of Theorem 3 closely mirrors the development in section 3 of [22]. The proof is by induction on t . To establish the base case $t = 0$, note that

$$\begin{aligned} S_0(z^{(0)}) &= K|\alpha| \sum_{j=1}^n (z_j^{(0)} - x_j^0)^2 + \Phi(x^0) + \langle g_0, z^{(0)} - x^0 \rangle \\ &\geq \frac{K|\alpha|}{2} \sum_{j=1}^n (z_j^{(0)} - x_j^0)^2 + \Phi(x^0) + \langle g_0, z^{(0)} - x^0 \rangle \\ (7) \quad &\geq \frac{K|\alpha|}{2} \sum_{j=1}^n (y_j^{(0)} - x_j^0)^2 + \Phi(x^0) + \langle g_0, y^{(0)} - x^0 \rangle \\ (8) \quad &\geq \Phi(y^{(0)}), \end{aligned}$$

where (7) follows from the definition of $y^{(0)}$ and (8) follows from Lemma 5.

To establish the induction step, assume that $\Phi(y^{(t)}) \leq S_t(z^{(t)})$. By definition, we have

$$\begin{aligned} S_{t+1}(x) &= \left(\frac{t+1}{t+3}\right) S_t(x) \\ &\quad + \left(\frac{2}{t+3}\right) [\Phi(x^{t+1}) + \langle g^{(t+1)}, x - x^{(t+1)} \rangle]. \end{aligned}$$

Since $\nabla^2 S_t = \frac{4K|\alpha|}{(t+1)(t+2)}I$, and S_t is minimized at z_t , we obtain

$$\begin{aligned}
 S_{t+1}(x) &\geq \left(\frac{t+1}{t+3}\right)S_t(z^{(t)}) + \frac{2K|\alpha|}{(t+2)(t+3)} \sum_j (x_j - z_j^{(t)})^2 \\
 (9) \qquad &+ \frac{2}{t+3} [\Phi(x^{(t+1)}) + \langle g^{(t+1)}, x - x^{(t+1)} \rangle].
 \end{aligned}$$

By the induction hypothesis and the convexity of Φ , it follows that $S_t(z^{(t)}) \geq \Phi(y^{(t)}) \geq \Phi(x^{(t+1)}) + \langle g^{(t+1)}, y^{(t)} - x^{(t+1)} \rangle$. Substituting this bound in (9) and rearranging terms, we get

$$\begin{aligned}
 S_{t+1}(x) &\geq \Phi(x^{(t+1)}) + \frac{2K|\alpha|}{(t+2)(t+3)} \sum_j (x_j - z_j^{(t)})^2 \\
 &+ \left\langle g^{(t+1)}, \left(\frac{2}{t+3}\right)x + \left(\frac{t+1}{t+3}\right)y^{(t)} - x^{(t+1)} \right\rangle \\
 &\geq \Phi(x^{(t+1)}) + \frac{2K|\alpha|}{(t+3)^2} \sum_j (x_j - z_j^{(t)})^2 \\
 &+ \left\langle g^{(t+1)}, \left(\frac{2}{t+3}\right)x + \left(\frac{t+1}{t+3}\right)y^{(t)} - x^{(t+1)} \right\rangle \\
 &= \Phi(x^{(t+1)}) \\
 &+ \frac{K|\alpha|}{2} \sum_j \left(\left(\frac{2}{t+3}\right)x_j + \left(\frac{t+1}{t+3}\right)y_j^{(t)} - x_j^{(t+1)} \right)^2 \\
 (10) \qquad &+ \left\langle g^{(t+1)}, \left(\frac{2}{t+3}\right)x + \left(\frac{t+1}{t+3}\right)y^{(t)} - x^{(t+1)} \right\rangle,
 \end{aligned}$$

where (10) is obtained by substituting $\left(\frac{2}{t+3}\right)z^{(t)} = x^{(t+1)} - \left(\frac{t+1}{t+3}\right)y^{(t)}$. Note that for $x \in P$, $\left(\frac{2}{t+3}\right)x + \left(\frac{t+1}{t+3}\right)y^{(t)} \in P$ as well. Thus, the expression in (10) is lower bounded by

$$S_{t+1}(x) \geq \Phi(x^{(t+1)}) + \min_{y \in P} \left\{ \frac{K|\alpha|}{2} \sum_j (y_j - x_j^{(t+1)})^2 + \langle g^{(t+1)}, y - x^{(t+1)} \rangle \right\}$$

$$(11) \qquad = \Phi(x^{(t+1)}) + \frac{K|\alpha|}{2} \sum_j (y_j^{t+1} - x_j^{(t+1)})^2 + \langle g^{(t+1)}, y^{t+1} - x^{(t+1)} \rangle$$

$$(12) \qquad \geq \Phi(y^{(t+1)}),$$

where (11) follows from the definition of $y^{(t+1)}$ and (12) follows from Lemma 5.

Remark. Note that the proofs of Corollary 4, Lemma 5, and Theorem 3 do not require $\alpha > 0$.

2.2.3. Notes on Algorithm QP. Having proved the correctness of Algorithm QP, we now give an intuitive motivation for the main algorithmic ideas behind it. The first step is to approximate the piecewise-linear function $\lambda(x)$ by the smooth potential function $\Phi(x)$. In all previous work on fractional packing problems, the next steps have been to construct the first-order Taylor series approximation to $\Phi(x)$ and minimize this linear approximation over the feasible set Q . The main bottleneck with the first-order methods is that the approximation is a *lower* bound; therefore,

one has to tightly control the step size to ensure that the improvement in the linear approximation is close to that in the potential function itself. As a direct consequence, the first-order methods yield algorithms whose complexity is $\mathcal{O}(\frac{1}{\epsilon^2})$.

The obvious next step is to consider the second-order approximation to $\Phi(x)$. Unfortunately, the Hessian of $\Phi(x)$ is dense and, therefore, the associated QP is relatively expensive. In [22], Nesterov proposed approximating the Hessian by its largest eigenvalue, i.e., replacing the Hessian with an identity matrix scaled by the largest eigenvalue. This immediately yields a second-order *upper* bound for $\Phi(x)$ that is easy to optimize. (An upper bound is superior to the lower bound since the improvement in $\Phi(x)$ is guaranteed to be more than the upper bound; consequently, one can set an aggressive step length.) Unfortunately, approximating the Hessian by its largest eigenvalue yields a very weak approximation, particularly when the iterate $x^{(t)}$ is far from the global minimum x^* . Algorithm QP corrects for this by computing two iterates: $y^{(t)}$ is a “local” iterate that uses only the “local” gradient information, i.e., the gradient at the current iterate $x^{(t)}$, and $z^{(t)}$ is a “global” iterate that uses a suitably weighted combination of the gradients at all the previous iterates. The next iterate, $x^{(t+1)}$, is a convex combination of the iterates $y^{(t)}$ and $z^{(t)}$. As iteration count t increases, the weight on the “local” iterate $y^{(t)}$ increases to 1, i.e., the algorithm becomes a Newton method, where the Hessian is approximated by its largest eigenvalue. The intuition of combining gradients at previous iterates to improve the approximation has also been previously proposed in the literature—Nesterov’s contribution showed that a particular combination leads to a worst-case bound on the complexity. It should be mentioned that this idea was also employed by Nemirovski [21] in a very different manner so as to obtain similar results.

Directly applying the technique proposed in [22] to the maximum concurrent flow problem does *not* yield a polynomial-time algorithm. Next, we discuss a particular scaling that we introduced in section 2.2 to construct a polynomial-time algorithm. When defining the set \bar{Q} in section 2.2 we impose the bounds $y_j \leq 1$ for all j . This bounding technique is essential in that it effectively reduces the width to at most n . This technique can be read in the approach used in [9] and [15]. More recent algorithms, such as those in [8] and [5], do not directly rely on this technique (arguably, their method for controlling width is *similar* in that they directly control the magnitudes of the solution variables) and potentially our overall scheme could be improved along the lines of [8] and [5]. The scaling technique in section 2.2 is essential for the analysis of Algorithm QP to go through; the scaling methods implicit in [9] and [15] will not work in this context.

2.3. Piecewise-linear approximation. Algorithm QP requires one to solve a sequence of separable QPs over P . In this section we describe a general method for approximating the separable convex quadratic function by piecewise-linear function with arbitrarily small error. This method is derived from one given in Minoux [20].

Fix $\sigma > 0$ and $w \geq 0$. Define a continuous convex piecewise-linear approximation $\mathcal{L}_{\sigma,w}(v)$ to the quadratic function $\frac{1}{2}(v - w)^2 : \mathbf{R}_+ \mapsto \mathbf{R}_+$ as

$$\mathcal{L}_{\sigma,w}(v) \triangleq \frac{1}{2}q^2\sigma^2 + \frac{w^2}{2} - wv + \left(q + \frac{1}{2}\right)\sigma(v - q\sigma), \quad v \in [q\sigma, (q + 1)\sigma), \quad q \in \mathbf{Z}_+.$$

For $q \in \mathbf{Z}_+$, the derivative $\mathcal{L}'_{\sigma,w}(q\sigma)$ is not defined. The *left-derivative* $\mathcal{L}^-_{\sigma,w}(q\sigma) = (q - \frac{1}{2})\sigma - w$ and the *right-derivative* $\mathcal{L}^+_{\sigma,w}(q\sigma) = (q + \frac{1}{2})\sigma - w$. For $v \in (q\sigma, (q + 1)\sigma)$, $q \in \mathbf{Z}$, the derivative $\mathcal{L}'_{\sigma,w}(v)$ exists and $\mathcal{L}'_{\sigma,w}(v) = \mathcal{L}^+_{\sigma,w}(v) = \mathcal{L}^-_{\sigma,w}(v) = (q + \frac{1}{2})\sigma - w$. The following properties are easy to obtain.

LEMMA 6. *The following hold for any $\sigma > 0$ and $w \in \mathbf{R}_+$.*

- (i) *For $q \in \mathbf{Z}_+$, $\mathcal{L}_{\sigma,w}(q\sigma) = \frac{1}{2}(q\sigma - w)^2$.*
- (ii) *For $v \in \mathbf{R}_+$, $\frac{1}{2}(v - w)^2 \leq \mathcal{L}_{\sigma,w}(v) \leq \frac{1}{2}(v - w)^2 + \frac{\sigma^2}{8}$.*
- (iii) *For $v \in \mathbf{R}_+$, $v - w - \frac{\sigma}{2} \leq \mathcal{L}_{\sigma,w}^-(v) \leq v - w \leq \mathcal{L}_{\sigma,w}^+(v) \leq v - w + \frac{\sigma}{2}$.*

2.4. Algorithm QP $_{\sigma}$. Algorithm QP $_{\sigma}$ approximates each convex quadratic term $\frac{1}{2}(x_j - \bar{x}_j)^2$ by the piecewise-linear $\mathcal{L}_{\sigma_j, \bar{x}_j}(x_j)$ with possibly different $\sigma_j \in (0, 1)$ for each variable x_j . Thus, each optimization problem in the course of Algorithm QP $_{\sigma}$ minimizes a piecewise-linear function over the convex set P .

ALGORITHM QP $_{\sigma}$

Input: $P \subseteq [0, 1]^n$, A , $\alpha = \frac{\ln m}{\gamma}$, $L = \gamma^{-1} \sqrt{16Kn \ln(m)}$, $\sigma_j \leq 2^{-p}$, $p = \lceil 3 \ln(L) \rceil = \lceil \frac{3}{2} \ln(16Kn \ln(m)) + 3 \ln(\frac{1}{\gamma}) \rceil$

Output: $\hat{y} \in P$ such that $\lambda(\hat{y}) \leq \lambda_P^* + \gamma$

choose $\hat{x}^0 \in P$. **set** $t \leftarrow 0$.

while ($t \leq L$) **do**

$g^{(t)} = \nabla \Phi(\hat{x}^{(t)})$

$\hat{y}^{(t)} \leftarrow \operatorname{argmin}_{x \in P} \left\{ K|\alpha| \sum_{j=1}^n \mathcal{L}_{\sigma_j, \hat{x}_j^{(t)}}(x_j) + \langle g^{(t)}, x - \hat{x}^{(t)} \rangle \right\}$

$\hat{S}_t(x) \leftarrow \frac{2}{t^2 + 3t + 2} \left\{ 2K|\alpha| \sum_{j=1}^n \mathcal{L}_{\sigma_j, \hat{x}_j^{(0)}}(x_j) + \sum_{h=0}^t (h+1) [\Phi(\hat{x}^h) + \langle g_h, x - \hat{x}^h \rangle] \right\}$

$\hat{z}^{(t)} \leftarrow \operatorname{argmin}_{x \in P} \{ \hat{S}_t(x) \}$

$\hat{x}^{(t+1)} \leftarrow \left(\frac{2}{t+3} \right) \hat{z}^{(t)} + \left(\frac{t+1}{t+3} \right) \hat{y}^{(t)}$

$t \leftarrow t + 1$

return $y^{(L)}$

In view of Lemma 6, we would expect that Algorithm QP $_{\sigma}$ successfully emulates Algorithm QP if the σ_j are small enough. In section 7.2 we provide a proof of the following fact.

THEOREM 7. *For any $t \geq 0$,*

$$\Phi(\hat{y}^{(t)}) \leq \hat{S}_t(\hat{z}^{(t)}) + \left(\frac{5K|\alpha|}{2} \right) \left(\sum_{h=1}^t \frac{1}{h^2} + t \right) \left(\sum_k \sigma_j \right).$$

This theorem implies the correctness of Algorithm QP $_{\sigma}$.

COROLLARY 8. *The output $y^{(L)}$ of Algorithm QP $_{\sigma}$ satisfies $\Phi(y^{(L)}) \leq \Phi^* + \gamma/2$.*

Proof. Suppose $\sigma_j \leq 2^{-p}$. Using Theorem 7 and Lemma 6(ii), we obtain

$$\begin{aligned} \Phi(\hat{y}^{(t)}) - \Phi^* &\leq \frac{2Kn|\alpha|}{(t+1)(t+2)} \left(1 + \frac{1}{4}(2^{-2p}) \right) \\ &\quad + \left(\frac{5K|\alpha|n}{2} \right) \left(\sum_{h=1}^t \frac{1}{h^2} + t \right) 2^{-p} \\ &< K|\alpha|n \left(\frac{2 + 2^{-(2p+1)}}{t^2} + \frac{5}{2}(2+t)2^{-p} \right). \end{aligned}$$

Suppose $t \geq 2$ and choose $p \geq 3 \ln t$. Then $\Phi(\hat{y}^{(t)}) - \Phi^* \leq \frac{8K|\alpha|n}{t^2}$. A simple calculation now establishes the result. \square

3. Concurrent flows with rational capacities and demands. In this section we discuss the special case of maximum concurrent flow with rational capacities and show that the piecewise-linear approximation introduced in section 2.3 can be solved efficiently for this special case.

Suppose we have a network $G = (V, E)$ with $|V|$ nodes, $|E|$ edges, and K commodities. We assume that the capacity u_e of every edge e is a positive rational. The demand vector d_k of every commodity k is also assumed to be a rational vector. Since scaling capacities and demands by a common positive constant does not change the value of the problem, we assume that all capacities and demands are integers. Let $f_{k,e}$ denote the flow associated with commodity k on edge e and let f_k denote the $|E|$ -vector with entries $f_{k,e}$. Then the maximum concurrent flow problem is given by

$$\begin{aligned} \lambda^* = \min \quad & \lambda \\ \text{s.t.} \quad & \sum_{k=1}^K f_{k,e} \leq \lambda u_e \quad \forall k, e, \\ & Nf_k = d_k, \quad f_k \geq 0, \quad k = 1, \dots, K, \end{aligned}$$

where N denotes the node-edge incidence matrix of the network. Let $\mathcal{F} = \{f : Nf_k = d_k, f_k \geq 0, k = 1, \dots, K\}$ denote the polyhedron of feasible flows.

In order to describe our piecewise-linear approach, in the following steps we review how the procedures we described in the prior sections would apply to the concurrent flow problem.

STEP 1. Define new, scaled variables $g_{k,e} = f_{k,e}/u_e$. This scaling leaves the objective unchanged, and the constraints are transformed into

$$\begin{aligned} \sum_k g_{k,e} &\leq \lambda, \quad e = 1, \dots, |E|, \\ g \in Q &= \{g \in \mathbf{R}^{K \times |E|} : \exists f \in \mathcal{F} \text{ with } g_{k,e} = f_{k,e}/u_e \forall k, e\}. \end{aligned}$$

The problem is now in the canonical form (1), with $m = |E|$ and $n = K|E|$.

STEP 2. After i iterations of BINARY SEARCH (see section 2.1), we get lower and upper bounds λ_l and λ_u with

$$(13) \quad \frac{\lambda_u - \lambda_l}{\lambda_l} \leq \left(\frac{2}{3}\right)^i O(\min\{m, K\}).$$

Let $\delta = \frac{1}{3}(\lambda_u - \lambda_l)$. We seek a vector g with $\max_e \sum_k g_{k,e} \leq \lambda^* + \delta$. Upon computing g , we reset either $\lambda_l \leftarrow \lambda_l + \delta$ or $\lambda_u \leftarrow \lambda_u - \delta$.

STEP 3. Let $P(\lambda_u) \triangleq \{z : \exists g \in Q \text{ with } z = g/\lambda_u, 0 \leq z \leq 1\}$. Procedure ABSOLUTE computes the vector g needed in Step 2 by approximately solving the scaled optimization problem

$$\begin{aligned} \min \quad & \lambda \\ \text{s.t.} \quad & \sum_{k=1}^K x_{k,e} \leq \lambda, \quad e = 1, \dots, |E|, \\ & x \in P(\lambda_u). \end{aligned}$$

The value of this problem is $\lambda^*/\lambda_u \leq 1$, and ABSOLUTE computes a feasible x with $\max_e \sum_k x_{k,e} \leq \lambda^*/\lambda_u + \gamma$, where $\gamma = \delta/\lambda_u$.

In section 2.2.1 we showed that in order to achieve the goal of procedure ABSOLUTE it is sufficient that x satisfy $\Phi(x) \leq \Phi^* + \gamma/2$. Corollary 8 in section 2.4 establishes that the output $\hat{y}^{(t)}$ produced by Algorithm QP_σ will satisfy this condition, provided

- (a) $\sigma_{k,e} \leq 2^{-p} \forall k, \forall e$,
- (b) $p \geq \bar{p} \triangleq \lceil \frac{3}{2} \ln(16Kn \ln(m)) + 3 \ln(\frac{1}{\gamma}) \rceil$,
- (c) $t \geq \frac{\sqrt{16Kn \ln m}}{\gamma}$.

Next we describe how to achieve (a)–(c) in the particular framework of the maximum concurrent flow problem. Note that in terms of the initial flow variables $f_{k,e}$, we have $x_{k,e} = \frac{1}{\lambda_u u_e} f_{k,e}$. We apply the framework developed in section 2.3 to the concurrent flow problem as follows:

- (i) We set $p = \bar{p} + \lceil \log D \rceil$, where D is the sum of all demands, and

$$(14) \quad \sigma_{k,e} = \frac{2^{-p}}{u_e} \quad \forall k, e.$$

This satisfies requirement (a) of Step 3.

- (ii) We modify BINARY SEARCH as follows. Every time a new upper bound λ_u is computed, we replace it with a *relaxed* bound $\hat{\lambda}_u \geq \lambda_u$, chosen so that $\frac{2^p}{\hat{\lambda}_u} = \lfloor \frac{2^p}{\lambda_u} \rfloor$. Note that $\lambda_u \leq D$, and so $\hat{\lambda}_u \leq \frac{\lambda_u}{1 - \lambda_u 2^{-p}} \leq \lambda_u (1 + O(2^{-p}))$. Since $\bar{p} = \frac{3}{2} \ln(Kn \ln(m)) + 3 \ln(\frac{1}{\gamma})$, where $\gamma = \frac{\delta}{\lambda_u}$, we have that

$$\frac{\hat{\lambda}^U - \lambda_l}{\lambda_u - \lambda_l} \leq 1 + \frac{\lambda_u O(2^{-\bar{p}})}{\lambda_u - \lambda_l} \leq 1 + O\left(\frac{2^{-\bar{p}}}{3\gamma}\right) = 1 + o(1).$$

Thus, up to constants, the complexity bound in Corollary 2 remains unchanged. For simplicity, in what follows we will use the notation λ_u to refer to the relaxed upper bound.

3.1. Solving the piecewise-linear problems. In this section we show how the modifications (i) and (ii) above allow one to efficiently solve the piecewise-linear problems encountered in Algorithm QP_σ .

The generic piecewise-linear problem that we need to solve is of the form

$$(15) \quad \begin{aligned} \min \quad & \sum_{k,e} \bar{\mathcal{L}}_{k,e}(x_{k,e}) \\ \text{s.t.} \quad & x \in P(\lambda_u), \end{aligned}$$

where $\bar{\mathcal{L}}_{k,e}(\cdot) = \mathcal{L}_{\sigma_{k,e}, \bar{x}_{k,e}}(\cdot)$ is a continuous convex piecewise-linear function with breakpoints at the integer multiples of $\frac{2^{-p}}{u_e}$ and with pieces of strictly increasing slope.

For every k and e , define $r_{k,e} = 2^p u_e x_{k,e}$. In terms of the initial flow variables $f_{k,e}$, we have $r_{k,e} = \frac{2^p}{\lambda_u} f_{k,e}$. Thus, after the change of variables the optimization problem is of the form

$$(16) \quad \begin{aligned} \min \quad & \sum_{k,e} \mathcal{L}_{k,e}(r_{k,e}) \\ \text{s.t.} \quad & Nr_k = \frac{2^p}{\lambda_u} d_k \quad \forall k, \\ & r_{k,e} \leq 2^p u_e \quad \forall k, e, \end{aligned}$$

where $\mathcal{L}_{k,e}$ is a continuous convex piecewise-linear function with breakpoints at the integers and pieces of strictly increasing slope. The optimization problem (16) is just a min-cost flow problem, with integral demands and capacities. In summary, as discussed in the previous section, given λ_u we will have to solve $2t$ problems of the form (16), where t is as in Step 3(c).

We solve each such problem using an approach similar to that described in [20] and in [1, Chap. 14]. This approach is reminiscent of the cost-scaling (or capacity-scaling) method for solving standard (linear) minimum-cost flow problems. Our algorithm is described in section 7.3. The algorithm assumes that a feasible, integral solution to (16) has been already computed (we will account for this work separately). Given such a solution, the algorithm solves problem (16) by performing

$$O\left(\sum_k \sum_e (p + \log u_e)\right) = O(K|E|p + KL_U)$$

shortest path computations, where L_U is the number of bits needed to represent the largest capacity. Since $p = \bar{p} + \lceil \log D \rceil$, over all iterations of Algorithm QP_σ the total number of shortest path computations incurred in calls to the algorithm in section 7.3 is

$$\begin{aligned} &O\left(\frac{\sqrt{Kn \ln(m)}}{\epsilon} (K|E|(\bar{p} + \lceil \log D \rceil) + KL_U)\right) \\ &= O^*\left(\epsilon^{-1} K^2 |E|^{\frac{1}{2}} \left(L_U + |E| \lceil \log D \rceil + |E| \log\left(\frac{1}{\epsilon}\right)\right)\right), \end{aligned}$$

plus lower-order complexity terms.

Finally, we account for the complexity of computing a feasible integral solution to (16). Note that all the problems of the form (16) arising in a given call to Algorithm QP_σ make use of the same value of λ_u and p ; consequently, the feasible integral flows need to be computed once per commodity, per call to QP_σ . In total, there will be $O^*(\epsilon^{-1})$ feasible flow computations per commodity. Each such computation essentially amounts to a maximum flow problem. We can solve such a problem using, e.g., any augmenting path algorithm (which essentially relies on shortest path computations). The simplest such algorithm (see [1]) performs $O(|V||E|)$ shortest path computations.

In summary, we now have the following result.

THEOREM 9. *An ϵ -optimal solution to a maximum concurrent flow problem, on a graph $G = (V, E)$ with $|V|$ nodes, $|E|$ edges, and K commodities, can be computed by solving*

$$O^*\left(\epsilon^{-1} K^2 E^{\frac{1}{2}} \left(L_U + |E| \lceil \log D \rceil + |E| \log\left(\frac{1}{\epsilon}\right)\right) + \epsilon^{-1} |V||E|\right)$$

shortest path problems, plus lower complexity steps, where L_U denotes the number of bits needed to store the capacities and D is the sum of demands.

4. Maximum multicommodity flows. In this section we extend the techniques developed in the previous section to the maximum multicommodity flow problem (see [2]; see also [8, 5]).

Let (s_k, t_k) , $k = 1, \dots, K$, denote a set of source-sink node pairs. The goal of the maximum multicommodity flow problem is to find a feasible multicommodity flow

(one that satisfies the capacity constraints on any edge) that maximizes the sum, over all k , of the amount of flow sent from s_k to t_k . Our approach will be to reduce this problem into an equivalent maximum concurrent flow problem.

Given a multicommodity flow, we will denote the total flow sent from s_k to t_k by F_k , $k = 1, \dots, K$. We denote by F^* the optimal value of the maximum multicommodity flow problem. Finally, let

$$(17) \quad F^L = \max_{k=1, \dots, K} \{\text{maxflow for commodity } k \text{ alone}\}.$$

Then $F^L \leq F^* \leq F^U = KF^L$. Moreover, for any feasible flow and any k , we have $F_k \leq F^L$. Suppose we introduce new variables $G_k = F^L - F_k \geq 0$. Then the maximum multicommodity flow problem is equivalent to

$$(18) \quad \begin{aligned} G^* \triangleq \min & \sum_{k=1}^K G_k \\ \text{s.t.} & F_k + G_k = F^L, \quad k = 1, \dots, K, \end{aligned}$$

$$(19) \quad \begin{aligned} & \sum_{k=1}^K f_{k,e} \leq u_e, \quad e \in E, \\ & f_k \text{ nonnegative and satisfies flow conservation,} \quad k = 1, \dots, K. \end{aligned}$$

Clearly, $\sum_k G_k \leq G^U \triangleq (K - 1)F^L$, and $F^* + G^* = F^U$. We assume that $F^L > 0$, or else the problem is trivial. Further, let $\delta = \min\{\frac{\epsilon}{K}, \frac{1}{2}\}$, and suppose $(\hat{f}_k, \hat{F}_k, \hat{G}_k)$, $k = 1, \dots, K$, is a vector satisfying

- (i) $\sum_{k=1}^K f_{k,e} \leq (1 + \delta)u_e \quad \forall e$,
- (ii) constraints (18) and (19), and
- (iii) $\sum_k \hat{G}_k \leq (1 + \delta)G^*$.

Then $\sum_k \hat{F}_k = F^U - \sum_k \hat{G}_k = F^* + G^* - \sum_k \hat{G}_k$, and consequently,

$$\frac{\sum_k \hat{F}_k}{F^*} \geq 1 - \delta \frac{G^*}{F^*} \geq 1 - \delta \frac{G^U}{F^L} = 1 - (K - 1)\delta.$$

As a result, the vector $((1 + \delta)^{-1}\hat{f}_k)$, $k = 1, \dots, K$, is an ϵ -optimal solution to the maximum multicommodity flow problem.

In order to find a vector satisfying conditions (i)–(iii), we first establish a nonzero lower bound for G^* . Consider the packing problem

$$(20) \quad \begin{aligned} \min & \max_{e \in E} \left\{ \frac{\sum_{k=1}^K f_{k,e}}{u_e} \right\} \\ \text{s.t.} & F_k = (1 - \delta/2)F^L \quad \forall k, \\ & f_k \text{ satisfies flow conservation} \quad \forall k. \end{aligned}$$

Note that (20) is a maximum concurrent flow problem. Let $\lambda(f)$ denote the maximum load of a flow f , and let \hat{f} denote any $(\delta/4)$ -optimal solution for (20). Consider the following two cases:

- (a) $\lambda(\hat{f}) \leq 1 + \delta/2$. In this case, $\hat{f}/(1 + \delta/2)$ is a feasible flow with total flow $(\sum_k F_k)/(1 + \delta/2) \geq (1 - \delta)KF^L = (1 - \delta)F^U$; i.e., $\hat{f}/(1 + \delta/2)$ is a δ -optimal multicommodity flow.

(b) $\lambda(\hat{f}) > 1 + \delta/2$. In this case, the packing problem (20) is infeasible. Thus, for any multicommodity flow satisfying the capacity constraints and flow conservation, we have $F_k < (1 - \delta/2)F^L$, i.e., $G^k > \delta F^L/2$ for at least one k .

Hence, $G^L \stackrel{\Delta}{=} \delta F^L/2$ is a valid lower bound for G^* .

We next describe the procedure that computes a vector satisfying (i)–(iii). The procedure maintains two bounds, $0 < G^L \leq G^* \leq G^U$, and a flow vector, \hat{f} . The bound G^L is initialized as indicated above and $G^U = (K - 1)F^L$. The flow vector \hat{f} is initialized as follows. Let \bar{k} denote the commodity attaining the maximum in (17). Then $\hat{f}_{\bar{k}}$ is set equal to any flow vector sending F^L units from $s_{\bar{k}}$ to $t_{\bar{k}}$, while for all other k , the flow vector \hat{f}_k is the zero vector. Thus, the flow \hat{f} attains G^U .

In any iteration of the procedure, we set $G = \frac{G^L + G^U}{2}$ and compute a $(\delta/8)$ -approximate solution to the packing problem

$$(21) \quad \lambda_G^* = \min \max \left\{ \max_{e \in E} \left\{ \frac{\sum_{k=1}^K f_{k,e}}{u_e} \right\}, \frac{\sum_k G_k}{G} \right\}$$

$$\text{s.t. } F_k + G_k = F^L \quad \forall k,$$

$$f_k \text{ satisfies flow conservation} \quad \forall k.$$

Note that (21) is a concurrent flow problem in the graph $G' = (V', E')$, $V' = V \cup \{S, T\}$, $E' = E \cup \{(S, T)\} \cup \{(s_k, S), (T, t_k) : k = 1, \dots, K\}$, $u_{(s_k, S)} = u_{(T, t_k)} = \infty$, and $u_{(S, T)} = G$ (the flows on edges (s_k, S) and (T, t_k) are both equal to G_k). Consequently, this task can be accomplished in $O^*(\frac{1}{\delta} \log(\frac{1}{\delta}))$ iterations of Algorithm QP $_{\sigma}$, where each iteration solves a piecewise-linear program that reduces to a sequence of shortest path problems.

Let λ_G denote the congestion of any $(\delta/8)$ -optimal solution. There are two cases as follows:

- (a) $\lambda_G > (1 + \delta/8)$. Then we reset to $G^L \leftarrow G$.
- (b) $\lambda_G \leq (1 + \delta/8)$. In this case we reset $G^U \leftarrow (1 + \delta/8)G$ and update \hat{f} to be the solution we computed to the packing problem.

The above procedure is repeated until $G^U - G^L \leq (\delta/2)G^L$. At the start of the procedure, $G^L = \delta F^L/2 \geq \delta G^U/2K$, and thus the number of bisection steps is bounded above by $O(\log(\frac{1}{\delta}))$ (neglecting polynomial factors in n , m , and K). To validate the procedure, note that each time that G^L is reset, i.e., case (a) holds, we have that $\lambda_G^* > 1$, which implies that $G \leq G^*$, and thus G^L is always a lower bound for G^* . When case (b) applies, the flow vector obtained in that iteration satisfies (i), (ii), and $\sum_k \hat{G}_k \leq (1 + \delta)G \leq (1 + \delta/8)G^U$. We conclude that at termination the vector \hat{f} is as desired.

In summary, the number of iterations of Algorithm QP $_{\sigma}$ needed to find an ϵ -optimal maximum multicommodity flow is bounded above by $O(\frac{1}{\epsilon} \log^2(\frac{1}{\epsilon}))$, again neglecting polynomial factors in n , m , and K .

5. Pure covering problems. The basic covering problem has the following structure:

$$(22) \quad \mu^* = \max_{x \in Q} \min_{1 \leq i \leq m} \{a_i^T x\},$$

where $Q \subseteq \mathbf{R}_+^n$ is a compact, convex set, $A = [a_1, \dots, a_m]^T \in \mathbf{R}_+^{m \times n}$, and each positive a_{ij} has value at least 1. For a given $x \in Q$, let $\mu(x) = \min_{1 \leq i \leq m} \{a_i^T x\}$. As in the packing problems, we will let K denote the maximum number of nonzero terms in any row of the A matrix.

5.1. Upper and lower bounds. In order to start our procedure, we need polynomially separated upper and lower bounds for μ^* . The following technique is used in [24]. For $i = 1, \dots, m$, define

$$(23) \quad \mu_i = \max_{x \in Q} \{a_i^T x\}, \quad \bar{x}_i = \operatorname{argmax}_{x \in Q} \{a_i^T x\}.$$

Then, it is clear that

$$(24) \quad \mu^* \leq \mu_u \triangleq \min_{1 \leq i \leq m} \{\mu_i\}.$$

Let $\bar{x} = \frac{1}{m} \sum_{k=1}^m \bar{x}_k$. Then

$$(25) \quad \begin{aligned} \mu(\bar{x}) &= \min_{1 \leq i \leq m} \{a_i^T \bar{x}\} = \frac{1}{m} \min_{1 \leq i \leq m} \left\{ \sum_{k=1}^m a_i^T \bar{x}_k \right\} \geq \frac{1}{m} \min_{1 \leq i \leq m} \{a_i^T \bar{x}_i\} = \frac{1}{m} \mu_u, \\ \mu_l &\triangleq \frac{1}{m} \mu_u \leq \mu^* \leq \mu_u. \end{aligned}$$

5.2. Refinement of bounds. In order to approximately compute μ^* , we would like to use a binary search procedure similar to that described in section 2.1. In the covering case, however, we need further elaboration. Given any $\mu_u > 0$, define

$$(26) \quad C(\mu_u) \triangleq \{y \in [0, 1]^n : \exists x \in Q \text{ s.t. } x \geq \mu_u y\}$$

and

$$(27) \quad \mu_u^* = \max_{y \in C(\mu_u)} \min_{1 \leq i \leq m} \{a_i^T y\}.$$

Then we have the following result.

LEMMA 10. *For any upper bound $\mu_u \geq \mu^*$, we have that $\mu_u^* = \frac{\mu^*}{\mu_u}$.*

Proof. Let $y^* = \operatorname{argmax}\{\mu(y) : y \in C(\mu_u)\}$. Since $y \in C(\mu_u)$, there exists $\hat{x} \in Q$ such that $\hat{x} \geq \mu_u y^*$. Thus,

$$\mu_u^* = \mu(y^*) \leq \frac{\mu(\hat{x})}{\mu_u} \leq \frac{\mu^*}{\mu_u}.$$

To prove the bound in the other direction, let $x^* = \operatorname{argmax}_{x \in Q} \mu(x)$. Define

$$\hat{y}_j = \frac{\min\{x_j^*, \mu_u\}}{\mu_u}, \quad j = 1, \dots, n.$$

Then $\hat{y} \in C(\mu_u)$, and to complete the proof it will suffice to show that, for any $1 \leq i \leq m$, $\sum_j a_{ij} \hat{y}_j \geq \frac{\mu^*}{\mu_u}$. To see that this is the case, fix i , and note that if $x_h^* > \mu_u$ for some h with $a_{ih} > 0$, then by the assumption on A , $\sum_j a_{ij} \hat{y}_j \geq \hat{y}_h = 1 \geq \frac{\mu^*}{\mu_u}$ (since $\mu_u \geq \mu^*$). If, on the other hand, $x_j^* \leq \mu_u$ for all j with $a_{ij} > 0$, then

$$\sum_j a_{ij} \hat{y}_j = \frac{\sum_j a_{ij} x_j^*}{\mu_u} \geq \frac{\mu^*}{\mu_u},$$

as desired. \square

The rest of the algorithm mirrors that for the packing case, in that we will use a potential function method to implement each step of the binary search. However, unlike in the packing case, the QPs solved in the course of approximately minimizing the potential function are defined over the sets $C(\mu_u)$, rather than over Q itself. At the end of this section we comment on why our approach needs the sets $C(\mu_u)$.

Consider a typical step of the binary search. We are given a lower bound μ_l and an upper bound μ_u on μ^* . Setting $\delta = \frac{\mu_u - \mu_l}{3}$, the outcome of the binary search step is an $x \in Q$ with $\mu(x) \geq \mu^* - \delta$; if $\mu(x) \geq \mu_l + 2\delta$, then we reset $\mu_l \leftarrow \mu_l + \delta$, while in the other case we reset $\mu_u \leftarrow \mu_l + 2\delta$, thereby improving the gap between the two bounds by a factor of $\frac{2}{3}$. In view of Lemma 10, we equivalently seek a $y \in C(\mu_u)$ with $\mu(y) \geq \mu_{P(\mu_u)}^* - \gamma$, where $\gamma = \frac{\delta}{\mu_u}$, and we will compare $\mu(y)$ with $\frac{\mu_l}{\mu_u} + 2\gamma$.

The computation of such a y is accomplished through the use of a potential function. For the covering case, we now use

$$\Phi(x) = \frac{1}{\beta} \ln \left(\sum_{i=1}^m e^{-\beta a_i^T x} \right)$$

for a suitably chosen $\beta > 0$. We have the following analogue of (5):

$$(28) \quad -\mu(x) \leq \Phi(x) \leq -\mu(x) + \frac{\ln m}{\beta}.$$

Thus, given $\gamma > 0$, in order to compute $y \in C(\mu_u)$ with $\mu(y) \geq \mu_u^* - \gamma$, we set $\beta = \frac{2 \ln m}{\gamma}$ and compute $y \in C(\mu_u)$ with $\Phi(y) \leq \Phi^* + \frac{\gamma}{2}$, where $\Phi^* \triangleq \min_{y \in P(\mu_u)} \Phi(y)$. For this purpose we can use Algorithm QP, given in section 2, verbatim. To see that this is a valid approach, recall that the proofs in section 2 that involve α and that of Lemma 5 are all valid for negative values of α . Thus, we have the following result.

THEOREM 11. *An ϵ -optimal solution to the covering problem can be computed by solving $O(\epsilon^{-1} \sqrt{Kn \ln m})$ convex separable QPs over the set $C(\mu_u)$ defined in (26).*

We conclude this section by listing some special cases, where convex separable QPs over the set $C(\mu_u)$ reduce to an optimization problem over Q or a simple modification of Q .

- (a) Q block angular: It is clear that $C(\mu_u)$ is also block angular. Moreover, $y \in C(\mu_u)$ if and only if there exists $s \geq 0$ such that $\mu_u y + s \in Q$. In other words, if we are given a linear inequality description of Q , then we obtain one for $C(\mu_u)$ with twice as many variables but with the same general structure.
- (b) Q “downwards closed”: Q is said to be “downwards closed” if $x \in Q$ implies that $\{x' : 0 \leq x' \leq x\} \subseteq Q$. Therefore, it follows that $C(\mu) = \{x \in Q : 0 \leq x_j \leq \mu \text{ for all } j\}$.
- (c) Q described by a linear optimization oracle: In this case, under appropriate assumptions, a convex QP over $C(\mu_u)$ reduces to a polynomial number of linear programs over Q [27]. The key ingredient here is that the separation problem over $C(\mu_u)$ can be solved by making a polynomial number of calls to the optimization oracle over Q . To see that this is the case, recall that we assumed $Q \subseteq \mathbf{R}_+^n$. Let

$$Q^{\leq}(\mu_u) = \{y \geq 0 : \exists x \in Q \text{ s.t. } x \geq \mu_u y\}.$$

Then, for any $c \in \mathbf{R}^n$, we have that

$$(29) \quad \max \left\{ \sum_j c_j y_j : y \in Q^{\leq}(\mu_u) \right\} = \max \left\{ \sum_j c_j^+ y_j : y \in Q^{\leq}(\mu_u) \right\},$$

where $r^+ \triangleq \max\{r, 0\}$. Since the objective vector on the right-hand side of (29) is nonnegative, it follows that this term is equal to

$$(\mu_u)^{-1} \max \left\{ \sum_j c_j^+ y_j : y \in Q \right\}.$$

Thus, a linear optimization problem over $Q^{\leq}(\mu_u)$ reduces to an equivalent linear optimization problem over Q . Consequently, the separation problem over $Q^{\leq}(\mu_u)$ requires a polynomial number of oracle calls (to the optimization oracle over Q). Since

$$C(\mu_u) = Q^{\leq}(\mu_u) \cap \{y \in R^n : y_j \leq 1 \ \forall j\},$$

we can conclude that the same holds for $C(\mu_u)$, as desired.

5.3. Why do covering problems require solving QPs over $C(\mu_u)$? In order to motivate our use of the sets $C(\mu_u)$, it is worth revisiting some of our techniques used for packing problems, in particular, the use of the potential function, which we adapted from the packing case in a straightforward manner in order to handle covering problems (essentially, by changing the sign of the exponent).

To understand the difference between packing and covering problems, let us examine the role of the sets P in packing problems. In the context of a fractional packing problem $\text{PACK}(A, Q)$ with $A \in \{0, 1\}^{m \times n}$, the critical observation is that $\lambda^* \leq \lambda_u$ implies that $x_j \leq \lambda_u$ for all $j = 1, \dots, n$. In this setting, the set P is given by $P = \lambda_u^{-1}Q \cap [0, 1]^n$. Next, consider a generalized packing problem with a nonnegative matrix A . We show that by introducing new variables this problem can be reduced to a packing problem $\text{PACK}(A, Q)$ with $A \in \{0, 1\}^{m \times n}$. The combined effect of adding variables and of scaling by λ_u^{-1} is the following: If $a_{ij} > 0$, then we generate a new variable of the form $\frac{a_{ij}}{\lambda_u} x_j$. The fact $\lambda^* \leq \lambda_u$ implies that, without loss of generality, we can place an upper bound of 1 on this new variable; equivalently, $\lambda^* \leq \lambda_u$ implies that the set $\bar{Q} = \{x \in Q : \frac{a_{ij}}{\lambda_u} x_j \leq 1 \text{ for all } a_{ij} > 0\}$ is nonempty, and $\lambda(x) \leq \lambda_u$ if and only if $x \in \bar{Q}$. For any $A \in \{0, 1\}^{m \times n}$ and $x \in P$, we have $\sum_j a_{ij} x_j \leq K$ for all i , where K denotes the maximum number of nonzero terms in any row of A . This bound is instrumental in the proofs of Corollary 4 and Lemma 5, both of which are critical in the analysis of the algorithm. In particular, the iteration count depends on K and not on the “width” [24] of the problem.

However, when dealing with a covering problem, this approach fails. Given an upper bound μ_u on μ^* , and a coefficient $a_{ij} > 0$, we cannot assume that $\frac{\mu_u}{a_{ij}}$ is an upper bound on x_j . The reason for this is that there might be a different row i' with $0 < a_{i'j} < a_{ij}$. Instead, we can assume only that $x_j \leq \frac{\mu_u}{a_{kj}}$, where a_{kj} is the smallest positive coefficient in column j of A . However, when we use this bound, the arguments in Corollary 4 and Lemma 5 do not follow through—instead of obtaining a dependence on K we obtain a dependence on the maximum ratio between positive entries in any given column j of A . Finally, we may be unable to place a (simple) upper bound on some x_j , even when $A \in \{0, 1\}^{m \times n}$ and there is a known upper bound $\mu_u \geq \mu^*$, because the structure of the set Q may simply prevent us from doing so—in other words, a set of the form $Q \cap \{x_j \leq \mu_u\}$ may be empty, in contrast with what happens in the packing case.

6. Algorithms for mixed packing and covering. In this section we consider the mixed packing-covering problem

$$(30) \quad \begin{aligned} \lambda^* \triangleq \min \quad & \lambda(x) \triangleq \max_{1 \leq i \leq m_p} \{p_i^T x\} \\ \text{s.t.} \quad & \min_{1 \leq i \leq m_c} \{c_i^T x\} \geq 1, \\ & x \geq 0, \end{aligned}$$

where $x \in \mathbf{R}^n$, and the vectors $p_i \in \mathbf{R}^n$, $i = 1, \dots, m_p$, and $c_i \in \mathbf{R}^n$, $i = 1, \dots, m_c$, are all assumed to be nonnegative. We present an algorithm that computes, for any $\epsilon > 0$, an ϵ -approximation to λ^* in $O^*(\frac{1}{\epsilon})$ steps. As before, each basic step consists of solving a separable, convex QP. As we will show, in this instance the QPs are particularly simple and can be solved in $O(nK_p)$ time.

6.1. Computing initial polynomially separated upper and lower bounds. The analysis in this section is identical to the one in [31]. Define $p = \frac{1}{m_p} \sum_{i=1}^{m_p} p_i$. For each $k = 1, \dots, m_c$, let

$$(31) \quad \lambda_k = \min \{p^T x : c_k^T x \geq 1, x \geq 0\} = \min_{1 \leq j \leq n} \left\{ \frac{p_j}{c_{kj}} \right\}.$$

Let $x^{(k)}$ denote the vector that achieves the minimum in (31), i.e.,

$$x_j^{(k)} = \begin{cases} \frac{1}{c_{kj}}, & j = \operatorname{argmin}_{1 \leq j' \leq n} \left\{ \frac{p_{j'}}{c_{kj'}} \right\}, \\ 0 & \text{otherwise.} \end{cases}$$

Then, it is clear that

$$\lambda^* \geq \lambda_l = \max_{1 \leq k \leq m_c} \{\lambda_k\}.$$

Let $\bar{x} = \sum_{k=1}^{m_c} x^{(k)}$. Then, for all $i = 1, \dots, m_c$,

$$c_i^T \bar{x} = \sum_{k=1}^{m_c} c_i^T x^{(k)} \geq c_i^T x^{(i)} \geq 1,$$

i.e., \bar{x} is feasible for (30), and

$$\max_{1 \leq i \leq m_p} \{p_i^T \bar{x}\} \leq \sum_{i=1}^{m_p} p_i^T \bar{x} = m_p p^T \bar{x} = m_p \sum_{k=1}^{m_c} \lambda_k \leq m_c m_p \lambda_l \triangleq \lambda_u.$$

Thus, we have a lower bound λ_l and an upper bound λ_u such that $\lambda_u \leq m_c m_p \lambda_l$.

6.2. Binary search to refine bounds. As with all other problems considered in this paper, we next refine the bounds using a binary search technique. In the general step of the binary search, we have the bounds $0 < \lambda_l \leq \lambda_u$. As before, set $\delta = \lambda_u - \lambda_l$ and $\gamma = \delta/\lambda_u$.

For $i = 1, \dots, m_p$, define $S_i = \{j : p_{ij} > 0\}$. Since $\lambda^* \leq \lambda_u$, it follows that for all $j \in S_i$ we can restrict $x_j \leq \lambda_u/p_{ij}$ without any loss in optimality. Thus, (30) is

equivalent to

$$(32) \quad \begin{aligned} \frac{\lambda^*}{\lambda_u} = \min \quad & \max_{1 \leq i \leq m_p} \left\{ \sum_{j \in S_i} y_{ij} \right\} \\ \text{s.t.} \quad & \sum_j c_{ij} x_j \geq 1, \quad i = 1, \dots, m_c, \\ & y_{ij} - \frac{p_{ij}}{\lambda_u} x_j = 0, \quad j \in S_i, \quad i = 1, \dots, m_p, \\ & y_{ij} \leq 1, \quad j \in S_i, \quad i = 1, \dots, m_p, \\ & x \geq 0. \end{aligned}$$

To refine the bounds we change variables by setting $z_{ij} = 1 - y_{ij}$ ($j \in S_i, i = 1, \dots, m_p$) and solve the following pure covering problem:

$$(33) \quad \begin{aligned} \mu^* = \max \quad & \mu \\ \text{s.t.} \quad & \sum_{j \in S_i} z_{ij} \geq \mu(|S_i| - (1 - 0.5\gamma)), \quad i = 1, \dots, m_p, \\ & \sum_j c_{ij} x_j \geq \mu, \quad i = 1, \dots, m_c, \\ & (x, z) \in Q, \end{aligned}$$

where the set Q consist of all pairs (x, z) satisfying

$$(34) \quad \begin{aligned} z_{ij} + \frac{p_{ij}}{\lambda_u} x_j &= 1, \quad j \in S_i, \quad i = 1, \dots, m_p, \\ z_{ij} &\geq 0, \quad j \in S_i, \quad i = 1, \dots, m_p, \\ x &\geq 0. \end{aligned}$$

Let $K_p = \max_{1 \leq i \leq m_p} \{|S_i|\}$, and let (\bar{x}, \bar{z}) denote any $(\frac{\gamma}{3K_p})$ -optimal solution for the pure covering problem (33). Consider the following two cases:

- (a) $\mu(\bar{x}, \bar{z}) < (1 - \frac{\gamma}{6K_p})$. In this case we claim that $1 - 0.5\gamma$ is a lower bound on $\frac{\lambda^*}{\lambda_u}$. Otherwise, there exists (\hat{x}, \hat{y}) feasible for (32) with $\sum_{j \in S_i} \hat{y}_{ij} \leq 1 - 0.5\gamma$ for each $1 \leq i \leq m_p$. Then it follows that $\mu^* \geq 1$, and a $(\frac{\gamma}{6K_p})$ -optimal solution to (33) must have $\mu \geq (1 - \frac{\gamma}{6K_p})$, a contradiction. Therefore $\lambda_l \leftarrow (1 - 0.5\gamma)\lambda_u$ is a valid lower bound for λ^* , and the gap $\delta = \lambda_u - \lambda_l$ decreases by half.
- (b) $\mu(\bar{x}, \bar{z}) \geq (1 - \frac{\gamma}{6K_p})$. Then $\sum_j c_{ij} \bar{x}_{ij} \geq 1 - \frac{\gamma}{6K_p}$, and

$$\sum_{j \in S_i} \bar{z}_{ij} \geq \left(1 - \frac{\gamma}{6K_p}\right) (|S_i| - (1 - 0.5\gamma)) \geq |S_i| - (1 - \gamma/3), \quad i = 1, \dots, m_p.$$

Define $\tilde{y}_{ij} = 1 - \bar{z}_{ij}, j \in S_i, i = 1, \dots, m_p$, and $(\tilde{x}, \tilde{y}) = \frac{1}{1 - \frac{\gamma}{6K_p}}(\bar{x}, \bar{y})$. Then it follows that $\sum_j c_{ij} \tilde{x}_{ij} \geq 1$ for all $i = 1, \dots, m_c$. Thus, (\tilde{x}, \tilde{y}) is feasible for (32). Further, for each $i = 1, \dots, m_p$, we have $\sum_{j \in S_i} \tilde{y}_{ij} \leq \frac{1 - \frac{\gamma}{3}}{1 - \frac{\gamma}{6K_p}} \leq 1 - \frac{\gamma}{6}$, or equivalently, $\sum_{j \in S_i} p_{ij} \tilde{x}_{ij} \leq (1 - \gamma/6)\lambda_u$. Consequently, $\lambda^* \leftarrow (1 - \gamma/6)\lambda_u$ is a valid upper bound, and the gap $\delta = \lambda_u - \lambda_l$ decreases by a factor of $5/6$.

In conclusion, in either case the gap is decreased by at least a factor of $5/6$. Therefore, it follows that we can compute an ϵ -optimal solution for the mixed packing-covering problem by solving $O(\ln(\epsilon^{-1} m_c m_p))$ covering problems of the form (33).

6.3. Overall complexity bound. In this section we compute a complexity bound for the pure covering problem (33) by exploiting the special structure of the set Q defined in (34).

Let $K_c = \max_{i=1}^{m_c} |\{j : c_{ij} > 0\}|$ denote the maximum number of nonzero entries in the covering constraints in the mixed packing-covering problem (30). Since the number of variables in the pure covering problem (33) is at most $n(1 + K_p)$, the number of constraints is $m_p + m_c$ and the maximum number of nonzero elements in the covering constraints is $\max\{K_p, K_c\}$, and $\gamma/K_p \geq \epsilon/n$. The results in section 5 imply that we can compute an $(\frac{\gamma}{3S})$ -optimal solution for (33) in

$$O\left(\frac{n}{\epsilon} \sqrt{\max\{K_c, K_p\}(1 + K_p)n \log(m_c + m_p)}\right)$$

iterations, where each iteration involves solving a convex separable QP over the set

$$C(\mu) = \left\{ (v, w) \in [0, 1]^{n+K_p} : \exists (x, z) \in Q \text{ s.t. } x \geq \mu v, z \geq \mu w \right\}.$$

See (26) for details. For Q defined in (34), the set $C(\mu)$ is given by

$$C(\mu) = \left\{ (v, w) \in [0, 1]^{n+K_p} : \mu v_j \leq \min \left\{ \min\{p_{ij}^{-1} \lambda_u : j \in S_i\}, 1 \right\}, \mu w_{ij} \leq (1 - \lambda_u \mu v_j / p_{ij}) \right\}.$$

A separable convex QP over $C(\mu)$ can be decomposed into n convex QPs in the variables (v, w) of the form

$$(35) \quad \begin{aligned} \min \quad & \alpha v - \frac{1}{2} \beta v^2 + \sum_{i:j \in S} \left(\eta_i w_i - \frac{1}{2} \kappa_i w_i^2 \right) \\ \text{s.t.} \quad & 0 \leq \mu w_i \leq (1 - \lambda_u \mu v / p_i), \quad i \in S, \\ & 0 \leq \mu v \leq \bar{v}. \end{aligned}$$

Fix v . Let $\bar{w}_i = \frac{\eta_i}{\kappa_i}$. If $\bar{w}_i \leq 0$, the optimal solution w_i^* of the one-dimensional QP in w_i variable is $w_i^* = 0$ and this variable can be removed from further consideration. When $\bar{w}_i > 0$, the optimal solution $w_i^* = \min\{\bar{w}_i, 1 - \frac{\lambda_u \mu v}{p_i}\}$. Thus, each index $i \in S$ partitions the interval $[0, \bar{v}/\mu]$ into two subintervals, one where $w_i^* = \bar{w}_i$ and the other where $w_i^* = 1 - \frac{\lambda_u \mu v}{p_i}$. Thus, the optimal value of the variable v can be computed in solving at most $|S| (\leq n)$ one-dimensional QPs. Thus, the computational effort required to compute an optimal solution of a convex separable QP over $C(\mu)$ is $O(nK_p)$. Combining this with our bound on the iteration count, we have the following result.

THEOREM 12. *The complexity of computing an ϵ -approximate solution for the mixed packing-covering problem (30) is*

$$O\left(\left(n^2 K_p \sqrt{\max\{K_c, K_p\}(1 + K_p)n \log(m_c + m_p)}\right) \frac{1}{\epsilon} \ln\left(\frac{m_c m_p}{\epsilon}\right)\right),$$

where K_p (resp., K_c) denote the maximum number of nonzeros in the packing (resp., covering) constraints.

7. Technical proofs.

7.1. Improved Taylor expansion. The derivative of $\nabla\Phi(x) = A^T\pi(x)$, where

$$\pi_i(x) = \frac{e^{\alpha a_i^T x}}{\sum_{j=1}^m e^{\alpha a_j^T x}}, \quad i = 1, \dots, m.$$

We want to show that $\|A^T(\pi(x) - \pi(y))\|_2 \leq K\alpha\|x - y\|_2$.

The construction of the bound will proceed in two steps as follows.

(a) \mathcal{L}_1 -bound on A^T : Since $\pi(x) \geq 0$ and $\sum_{i=1}^m \pi_i(x) = 1$ for all x , we endow the π -space with the \mathcal{L}_1 -norm. Define $\|A\|_{1,2}$ as

$$\begin{aligned} \|A\|_{1,2} &= \max \{ \|A^T\pi\|_2 : \|\pi\|_1 = 1 \} \\ &= \max \left\{ \left\| \sum_{i=1}^m \pi_i a_i \right\|_2 : \|\pi\|_1 = 1 \right\} \\ &= \max_{1 \leq i \leq m} \{ \|a_i\|_2 \} = \sqrt{K}, \end{aligned}$$

where the first equality in the last step follows from the fact that $\|a\|_2$ is a convex function and achieves its maximum at the extreme points of the feasible set, and the final bound follows from the fact that number of 1's in the vectors a_i is bounded above by K . Thus, we have that

$$(36) \quad \begin{aligned} \|A^T(\pi(x) - \pi(y))\|_2^2 &\leq \|A\|_{1,2}^2 \|\pi(x) - \pi(y)\|_1^2 \\ &\leq K \|\pi(x) - \pi(y)\|_1^2. \end{aligned}$$

(b) Strong convexity of the entropy function: The next step is to show an upper bound of the form $\|\pi(x) - \pi(y)\|_1 \leq C\|x - y\|_2$ for an appropriate C . Since we need an upper bound on a norm, one way to achieve this is to bound it above by a strongly convex function.

Define $H(\pi) = \sum_{i=1}^m \pi_i \log(\pi_i)$ for π such that $\pi \geq 0$ and $\sum_{i=1}^m \pi_i = 1$. Then, $\nabla^2 H = \text{diag}(1/\pi_i)$, and

$$y^T(\nabla^2 H)y = \sum_{i=1}^m \frac{y_i^2}{\pi_i}.$$

The right-hand side is a convex function and, on minimizing this function over π in the simplex, we obtain that

$$(37) \quad y^T(\nabla^2 H)y \geq \left(\sum_{i=1}^m |y_i| \right)^2.$$

Thus, we have that

$$\begin{aligned} &(\nabla H(\pi(x)) - \nabla H(\pi(y)))^T (\pi(x) - \pi(y)) \\ &= (\nabla^2 H(\pi_\theta)(\pi(x) - \pi(y)))^T (\pi(x) - \pi(y)) \geq \|\pi(x) - \pi(y)\|_1^2, \end{aligned}$$

where $\pi_\theta = \theta\pi(x) + (1 - \theta)\pi(y)$, $\theta \in [0, 1]$, the first equality follows from the

mean value theorem, and the second inequality follows from (37). Substituting the values of $\nabla H(\pi(x))$ and $\nabla H(\pi(y))$, it follows that

$$\begin{aligned}
 \|\pi(x) - \pi(y)\|_1^2 &\leq \sum_{i=1}^m (\log(\pi_i(x)) - \log(\pi_i(y))) (\pi_i(x) - \pi_i(y)) \\
 &= \sum_{i=1}^m \left(\alpha(a_i^T x) - \log \left(\sum_j e^{\alpha a_j^T x} \right) \right) (\pi_i(x) - \pi_i(y)) \\
 &\quad - \sum_{i=1}^m \left(\alpha(a_i^T y) + \log \left(\sum_j e^{\alpha a_j^T y} \right) \right) (\pi_i(x) - \pi_i(y)) \\
 &= \alpha \sum_{i=1}^m (a_i^T(x - y)) (\pi_i(x) - \pi_i(y)) \\
 &\quad - \log \left(\sum_j e^{\alpha a_j^T y} \right) \sum_{i=1}^m (\pi_i(x) - \pi_i(y)) \\
 (38) \quad &\quad + \log \left(\sum_j e^{\alpha a_j^T x} \right) \sum_{i=1}^m (\pi_i(x) - \pi_i(y)) \\
 &= \alpha \sum_{i=1}^m (a_i^T(x - y)) (\pi_i(x) - \pi_i(y)) \\
 &= \alpha(x - y)^T (A^T(\pi(x) - \pi(y))) \\
 (39) \quad &\leq |\alpha| \|x - y\|_2 \|A^T(\pi(x) - \pi(y))\|_2,
 \end{aligned}$$

where (38) follows from the fact that $\pi(x), \pi(y)$ are both elements of the simplex, and (39) follows from the Cauchy–Schwartz inequality.

Combining (36) and (39) we get

$$\begin{aligned}
 \|A^T(\pi(x) - \pi(y))\|_2 &\leq |\alpha| \|A\|_{1,2}^2 \|x - y\|_2, \\
 &\leq K |\alpha| \|x - y\|_2.
 \end{aligned}$$

7.2. Proof of Theorem 7. We begin with the approximate version of the Taylor expansion (see Lemma 5).

LEMMA 13. *At any iteration t of Algorithm QP_σ , we have that for any $x \in P$,*

$$\hat{S}_t(x) - \hat{S}_t(\hat{z}^{(t)}) \geq \frac{2K|\alpha|}{(t+1)(t+2)} \left\{ \sum_{j=1}^n (x_j - \hat{z}_j^{(t)})^2 - \sum_{j=1}^n \left(\sigma_j + \frac{1}{4}\sigma_j^2 \right) \right\}.$$

Proof. Fix $x \in P$. Recall that the function $\hat{S}_t(x)$ is defined as

$$\hat{S}_t(x) = \frac{2}{(t+1)(t+2)} \left\{ 2K|\alpha| \sum_{j=1}^n \mathcal{L}_{\sigma_j, \hat{x}_j^{(0)}}(x_j) + \sum_{h=0}^t (h+1) [\Phi(\hat{x}^h) + \langle g_h, x - \hat{x}^h \rangle] \right\}.$$

Define

$$\bar{S}_t(x) = \frac{2}{(t+1)(t+2)} \left\{ K|\alpha| \sum_{j=1}^n (x_j - \hat{x}_j^{(0)})^2 + \sum_{h=0}^t (h+1) [\Phi(\hat{x}^{(h)}) + \langle \hat{g}^{(h)}, x - \hat{x}^{(h)} \rangle] \right\}.$$

Then, by Lemma 6(ii),

$$(40) \quad \hat{S}_t(x) - \hat{S}_t(\hat{z}^{(t)}) \geq \bar{S}_t(x) - \bar{S}_t(\hat{z}^{(t)}) - \frac{2K|\alpha|}{(t+1)(t+2)} \left\{ \sum_{j=1}^n \frac{1}{4} \sigma_j^2 \right\}.$$

Since \bar{S}_t is a quadratic function, it follows that

$$(41) \quad \bar{S}_t(x) - \bar{S}_t(\hat{z}^{(t)}) = \frac{2K|\alpha|}{(t+1)(t+2)} \sum_{j=1}^n (x_j - \hat{z}_j^{(t)})^2 + \langle \nabla \bar{S}_t(\hat{z}^{(t)}), x - \hat{z}^{(t)} \rangle.$$

Consider the function \hat{S}_t restricted to the one-dimensional segment between $\hat{z}^{(t)}$ and x . \hat{S}_t is convex, piecewise-linear, and minimized at $\hat{z}^{(t)}$ (by definition of $\hat{z}^{(t)}$). Hence, as we traverse the segment from $\hat{z}^{(t)}$ to x , the slope of the first piece of the piecewise-linear function must be nonnegative. In other words,

$$(42) \quad \frac{2}{(t+1)(t+2)} \left\{ \sum_{j=1}^n \left[K|\alpha| \lambda_j + \sum_{h=0}^t (h+1) g_j^{(h)} \right] (x_j - \hat{z}_j^{(t)}) \right\} \geq 0,$$

where for $1 \leq j \leq n$,

$$\lambda_j = \begin{cases} \mathcal{L}_{\sigma_j, \hat{x}_j^{(0)}}^+(\hat{z}_j^{(t)}), & x_j \geq \hat{z}_j^{(t)}, \\ \mathcal{L}_{\sigma_j, \hat{x}_j^{(0)}}^-(\hat{z}_j^{(t)}), & \text{otherwise,} \end{cases}$$

and $g_j^{(h)}$ is the j th coordinate of $g^{(h)}$, $j = 0, \dots, t$. Since $P \subseteq [0, 1]^n$, by Lemma 6(iii) the second term in the right-hand side of (41) is at least $-\frac{2K|\alpha|}{(t+1)(t+2)} \sum_{j=1}^n \sigma_j$; consequently,

$$\bar{S}_t(x) - \bar{S}_t(\hat{z}^{(t)}) \geq \frac{2K|\alpha|}{(t+1)(t+2)} \left\{ \sum_{j=1}^n (x_j - \hat{z}_j^{(t)})^2 - \sum_{j=1}^n \sigma_j \right\}.$$

Together with equation (40) this implies the desired result. \square

Theorem 7 is established by induction on t . By definition, we have that

$$\begin{aligned} \hat{S}_0(\hat{z}^{(0)}) &= 2K|\alpha| \sum_{j=1}^n \mathcal{L}_{\sigma_j, \hat{x}_j^{(0)}}(\hat{z}_j^{(0)}) + \Phi(\hat{x}^0) + \langle \hat{g}^{(0)}, z^{(0)} - x^0 \rangle \\ &\geq K|\alpha| \sum_{j=1}^n \mathcal{L}_{\sigma_j, \hat{x}_j^0}(\hat{z}_j^{(0)}) + \Phi(\hat{x}^0) + \langle \hat{g}^{(0)}, z^{(0)} - x^0 \rangle \\ &\geq K|\alpha| \sum_{j=1}^n \mathcal{L}_{\sigma_j, \hat{x}_j^0}(\hat{y}_j^{(0)}) + \Phi(\hat{x}^0) + \langle \hat{g}^{(0)}, y^{(0)} - x^0 \rangle \\ (43) \quad &\geq \frac{K|\alpha|}{2} \sum_{j=1}^n (y_j^{(0)} - x_j^0)^2 + \Phi(x^0) + \langle g^{(0)}, y^{(0)} - x^0 \rangle \\ (44) \quad &\geq \Phi(y^{(0)}), \end{aligned}$$

where (43) follows from Lemma 6(ii) and (44) follows from the definition of $y^{(0)}$.

Next, we establish the inductive step. Let $x \in P$. By Lemma 13, we have

$$\begin{aligned} \hat{S}_t(x) &\geq \hat{S}_t(\hat{z}^{(t)}) + \frac{2K|\alpha|}{(t+1)(t+2)} \left\{ \sum_{j=1}^n (\hat{z}_j^{(t)} - \hat{x}_j^0)^2 - \sum_{j=1}^n \left(\sigma_j + \frac{1}{4}\sigma_j^2 \right) \right\} \\ &\geq \hat{S}_t(\hat{z}^{(t)}) + \frac{2K|\alpha|}{(t+1)(t+2)} \sum_{j=1}^n (\hat{z}_j^{(t)} - \hat{x}_j^0)^2 - \frac{5K|\alpha|}{2(t+1)^2} \left(\sum_{j=1}^n \sigma_j \right). \end{aligned}$$

Applying the induction hypothesis, and continuing as in the proof of Theorem 3, we obtain the following analogue of the inequality following (11):

$$\begin{aligned} \hat{S}_{t+1}(x) &\geq \Phi(\hat{x}^{(t+1)}) \\ &\quad + \min_{y \in P} \left\{ \frac{K|\alpha|}{2} \sum_j (y_j - \hat{x}_j^{(t+1)})^2 + \langle \hat{g}^{(t+1)}, y - \hat{x}^{(t+1)} \rangle \right\} \\ &\quad - \left(\frac{5K|\alpha|}{2} \right) \left(\sum_{h=1}^{t+1} \frac{1}{h^2} + t \right) \left(\sum_{j=1}^n \sigma_j \right). \end{aligned}$$

Applying Lemma 6 again, we obtain

$$\begin{aligned} \hat{S}_{t+1}(x) &\geq \Phi(\hat{x}^{(t+1)}) + K|\alpha| \sum_j \mathcal{L}_{\sigma_j, \hat{x}_j^{(t+1)}}(\hat{y}_j^{t+1}) + \langle \hat{g}^{(t+1)}, \hat{y}^{t+1} - \hat{x}^{(t+1)} \rangle \\ &\quad - \left(\frac{5K|\alpha|}{2} \right) \left(\sum_{h=1}^{t+1} \frac{1}{h^2} + t + 1 \right) \left(\sum_{j=1}^n \sigma_j \right) \\ &\geq \Phi(\hat{x}^{(t+1)}) + \frac{K|\alpha|}{2} \sum_j (\hat{y}_j^{t+1} - \hat{x}_j^{(t+1)})^2 + \langle \hat{g}^{(t+1)}, \hat{y}^{t+1} - \hat{x}^{(t+1)} \rangle \\ &\quad - \left(\frac{5K|\alpha|}{2} \right) \left(\sum_{h=1}^{t+1} \frac{1}{h^2} + t + 1 \right) \left(\sum_{j=1}^n \sigma_j \right) \\ &\geq \Phi(\hat{y}^{t+1}) - \left(\frac{5K|\alpha|}{2} \right) \left(\sum_{h=1}^{t+1} \frac{1}{h^2} + t + 1 \right) \left(\sum_{j=1}^n \sigma_j \right), \end{aligned}$$

where the last inequality follows from Lemma 5.

7.3. Piecewise-linear min-cost flow problems. We are given an optimization problem of the form described in section 3.1,

$$(45) \quad \begin{aligned} \min \quad &\sum_{k,e} \mathcal{L}_{k,e}(r_{k,e}) \\ \text{s.t.} \quad &Nr_k = \hat{d}_k, \quad 0 \leq r_k \leq \hat{u}_k, \quad k = 1, \dots, K, \end{aligned}$$

where, for every k and e , $\mathcal{L}_{k,e}$ is a continuous, convex, piecewise-linear function with breakpoints at the integers and with pieces of strictly increasing slope; \hat{d}_k and \hat{u}_k are integral; and $u_{k,e} \leq 2^q$ for an appropriate integer $q > 0$. For each k and each vertex i , denote by $\hat{d}_{k,i}$ the i th component of \hat{d}_k .

We assume that for each k we have an integral flow that satisfies the constraints of (45). Thus, we can convert (45) into an equivalent *circulation form*. Then we will have a problem of the form

$$(46) \quad \begin{aligned} \min \quad & \sum_{k,e} \mathcal{L}_{k,e}(r_{k,e}) \\ \text{s.t.} \quad & Nr_k = 0, \quad -a_k \leq r_k \leq b_k, \quad k = 1, \dots, K, \end{aligned}$$

where, for every k and e , the parameters $a_{k,e}$ and $b_{k,e}$ are integral and satisfy $0 \leq a_{k,e}, b_{k,e} \leq 2^q$, and again $\mathcal{L}_{k,e}$ is a convex, continuous, piecewise-linear function with breakpoints at the integers (the $\mathcal{L}_{k,e}$ in (46) equal those in (45), shifted by appropriate amounts).

We solve (46) by solving a sequence of circulation problems—our approach is similar to [20]. For all integers $h \in [0, q]$, $k = 1, \dots, K$, and $e \in E$, let $\mathcal{L}_{k,e}^{(h)}$ denote the continuous, convex, piecewise-linear function that has breakpoints at the integer multiples of 2^h and that agrees with $\mathcal{L}_{k,e}$ at each breakpoint.

Further, define $a_{k,e}^{(h)} = \lceil 2^{-h} a_{k,e} \rceil$ and $b_{k,e}^{(h)} = \lceil 2^{-h} b_{k,e} \rceil$. Then, the *level- h* problem is

$$(47) \quad \begin{aligned} \min \quad & \sum_{k,e} \mathcal{L}_{k,e}^{(h)}(r_{k,e}) \\ \text{s.t.} \quad & Nr_k = 0, \quad -b_k^{(h)} \leq r_k \leq a_k^{(h)} \quad \forall k. \end{aligned}$$

Thus, the level-0 problem is (46). We solve it by first solving the level- q problem, then the level- $(q - 1)$ problem, and so on. Note that for $0 \leq h \leq q$, the function $\mathcal{L}_{k,e}^{(h)}$ has 2^{q-h} breakpoints in the range of the level- h problem. Hence, the level- h problem can be seen as an ordinary (e.g., linear) minimum-cost circulation problem, on the graph $\hat{G}^{(h)}$ obtained from the original graph G , by replacing each edge e with 2^{q-h} parallel arcs, each with capacity 2^h and appropriate cost. (To avoid confusion, we use the term *arc*, rather than *edge*, which we reserve for G .) We stress that our algorithm will *only implicitly* work with \hat{G}^h .

Inductively, suppose we have solved the level- h problem. We can assume, without loss of generality, that all the entries of optimal circulation $r_k^{(h)}$ are integer multiples of 2^h . Let π_k^h denote the node potentials (see [20] or [1] for details). Our task is to refine $r_k^{(h)}$ into an optimal circulation for the level- $(h - 1)$ problem.

Note that by definition, for any k and e ,

- (a) the functions $\mathcal{L}_{k,e}^{(h)}$ and $\mathcal{L}_{k,e}^{(h-1)}$ agree at integer multiples of 2^{h-1} ,
- (b) we let $q \in \mathbf{Z}_+$. Then the slope of $\mathcal{L}_{k,e}^{(h-1)}$ is less (resp., more) than the slope of $\mathcal{L}_{k,e}^{(h)}$ in the interval $[2^h q, 2^h q + 2^{h-1})$ (resp., in the interval $[2^h q + 2^{h-1}, 2^h(q + 1))$),
- (c) either $a_{k,e}^{(h-1)} = a_{k,e}^{(h)}$ or $a_{k,e}^{(h-1)} = a_{k,e}^{(h)} - 2^{h-1}$, and similarly with the pair $(b_{k,e}^{(h-1)}, b_{k,e}^h)$.

Thus, it is easy to see that $r_k^{(h)}$, together with the potentials $\pi_k^{(h)}$, *nearly* satisfies the optimality conditions for the level- $(h - 1)$ problem. More precisely, suppose we convert $r_{k,e}^h$ into a circulation on the graph $\hat{G}^{(h-1)}$ by following the “greedy” rule: for any k and e , we “fill” the parallel arcs corresponding to k, e in increasing order of cost (and thus, at most one arc will have flows strictly between bounds). We may need an additional, “overflow” arc, also of capacity 2^{h-1} , in the case that $r_{k,e}^{(h)} = a_{k,e}^{(h)} > a_{k,e}^{(h-1)}$ and similarly for the case when $r_{k,e}^{(h)} = b_{k,e}^{(h)}$.

Denote by $\hat{r}_{k,e}^{(h)}$ the resulting circulation in \hat{G}^{h-1} . Then by properties (a)–(c) above, it follows that at most *one* of the parallel arcs corresponding to a pair (k, e) either fails to satisfy the optimality conditions with respect to the potentials $\pi_k^{(h)}$ or is an overflow arc. Consequently, we can obtain an optimal circulation in $\hat{G}^{(h-1)}$ in at most $O(|E|)$ flow pushes (each pushing $2^{(h-1)}$ units of flow) or computations of node potentials, and each such step requires the solution of a shortest path problem. It is clear that (again because of (a)–(c) above) all of this can be done without explicitly considering the graph $\hat{G}^{(h-1)}$: instead, we always keep a single flow value for commodity k on any edge e , which is always an integral multiple of $2^{(h-1)}$; if we wish to use one of the parallel arcs corresponding to k, e in a push (or when searching for an augmenting path), then it takes $O(1)$ time to determine *which* of the arcs we will use. This completes the description of the inductive step.

In summary, we have the following.

LEMMA 14. *Problem (45) can be solved by performing $O(\sum_k \sum_e \log u_{k,e})$ shortest path computations.*

Acknowledgments. We thank the anonymous referees and L. Babai for helping us improve the presentation of this paper.

REFERENCES

- [1] R. AHUJA, T. L. MAGNANTI, AND J. ORLIN, *Networks Flows: Theory, Algorithms, and Practice*, Prentice–Hall, Englewood Cliffs, NJ, 1993.
- [2] B. AWERBUCH AND T. LEIGHTON, *A simple local-control approximation algorithm for multi-commodity flow*, in Proceedings of the 34th Annual IEEE Symposium on Foundations of Computer Science, IEEE, Piscataway, NJ, 1993, pp. 459–468.
- [3] D. BIENSTOCK, *Potential Function Methods for Approximately Solving Linear Programming Problems: Theory and Practice*, Kluwer, Boston, MA, 2002.
- [4] D. BIENSTOCK AND O. RASKIN, *Asymptotic analysis of the flow deviation method for the maximum concurrent flow problem*, Math. Prog., 91 (2002), pp. 379–492.
- [5] L. K. FLEISCHER, *Approximating fractional multicommodity flow independent of the number of commodities*, SIAM J. Discrete Math., 13 (2000), pp. 505–520.
- [6] M. FRANK AND P. WOLFE, *An algorithm for quadratic programming*, Naval Res. Logist. Quart., 3 (1956), pp. 95–110.
- [7] L. FRATTA, M. GERLA, AND L. KLEINROCK, *The flow deviation method: An approach to store-and-forward communication network design*, Networks, 3 (1971), pp. 97–133.
- [8] N. GARG AND J. KÖNEMANN, *Faster and simpler algorithms for multicommodity flow and other fractional packing problems*, in Proceedings of the 39th Annual IEEE Symposium on Foundations of Computer Science, IEEE, Piscataway, NJ, 1998, pp. 300–309.
- [9] M. D. GRIGORIADIS AND L. G. KHACHYAN, *Fast approximation schemes for convex programs with many blocks and coupling constraints*, SIAM J. Optim., 4 (1994), pp. 86–107.
- [10] M. D. GRIGORIADIS AND L. G. KHACHYAN, *An exponential-function reduction method for block-angular convex programs*, Networks, 26 (1995), pp. 59–68.
- [11] M. D. GRIGORIADIS AND L. G. KHACHYAN, *Approximate minimum-cost multicommodity flows in $\tilde{O}(\epsilon^{-2}KNM)$ time*, Math. Prog., 75 (1996), pp. 477–482.
- [12] M. GRÖTSCHEL, L. LOVÁSZ, AND A. SCHRIJVER, *The ellipsoid method and its consequences in combinatorial optimization*, Combinatorica, 1 (1981), pp. 169–197.
- [13] M. GRÖTSCHEL, L. LOVÁSZ, AND A. SCHRIJVER, *Geometric Algorithms and Combinatorial Optimization*, 2nd ed., Springer-Verlag, Berlin, 1993.
- [14] S. KAPOOR AND P. M. VAIDYA, *Fast algorithms for convex quadratic programming and multi-commodity flows*, in Proceedings of the 18th Annual ACM Symposium on Theory of Computing, ACM, New York, 1986, pp. 147–159.
- [15] P. KLEIN, S. PLOTKIN, C. STEIN, AND É. TARDOS, *Faster approximation algorithms for the unit capacity concurrent flow problem with applications to routing and finding sparse cuts*, in Proceedings of the 22nd Annual ACM Symposium on Theory of Computing, ACM, New York, 1990, pp. 310–321.

- [16] P. KLEIN AND N. YOUNG, *On the number of iterations for Dantzig–Wolfe optimization and packing-covering approximation algorithms*, in *Integer Programming and Combinatorial Optimization*, Springer, Berlin, 1999, pp. 320–327.
- [17] M. K. KOZLOV, S. P. TARASOV, AND L. G. KHACHIYAN, *Polynomial solvability of convex quadratic programming*, *Soviet Math. Dokl.*, 20 (1979), pp. 1108–1111.
- [18] M. K. KOZLOV, S. P. TARASOV, AND L. G. KHACHIYAN, *The polynomial solvability of convex quadratic programming*, *USSR Comput. Math. Phys.*, 20 (1980), pp. 223–228.
- [19] T. LEIGHTON, F. MAKEDON, S. PLOTKIN, C. STEIN, É. TARDOS, AND S. TRAGOUDAS, *Fast approximation algorithms for multicommodity flow problems*, in *Proceedings of the 23rd Annual ACM Symposium on Theory of Computing*, ACM, New York, 1999, pp. 101–111.
- [20] M. MINOUX, *A polynomial algorithm for minimum quadratic cost flows*, *European J. Oper. Res.*, 18 (1984), pp. 377–387.
- [21] A. NEMIROVSKI, *Prox-method with rate of convergence $O(1/t)$ for variational inequalities with Lipschitz continuous monotone operators and smooth convex-concave saddle point problems*, *SIAM J. Optim.*, 15 (2004), pp. 229–251.
- [22] Y. NESTEROV, *Smooth minimization of non-smooth functions*, *Math. Program. Ser. A*, 103 (2005), pp. 127–152.
- [23] S. PLOTKIN AND D. KARGER, *Adding multiple cost constraints to combinatorial optimization problems, with applications to multicommodity flows*, in *Proceedings of the 27th Annual ACM Symposium on Theory of Computing*, ACM, New York, 1995, pp. 18–25. Available online at <http://troll-w.stanford.edu/plotkin/mcf.html>.
- [24] S. PLOTKIN, D. B. SHMOYS, AND É. TARDOS, *Fast approximation algorithms for fractional packing and covering problems*, *Math. Oper. Res.*, 20 (1995), pp. 495–504.
- [25] T. RADZIK, *Fast deterministic approximation for the multicommodity flow problem* in *Proceedings of the 6th Annual ACM-SIAM Symposium on Discrete Algorithms*, ACM, New York, SIAM, Philadelphia, pp. 486–492.
- [26] F. SHAHROKHI AND D. W. MATULA, *The maximum concurrent flow problem*, *J. ACM*, 37 (1991), pp. 318–334.
- [27] L. TUNÇEL, *Polyhedral and Semidefinite Programming Methods in Combinatorial Optimization*, Lecture Notes, Faculty of Mathematics, University of Waterloo, Waterloo, Ontario, Canada. Available from the author.
- [28] P. M. VAIDYA, *Speeding up linear programming using fast matrix multiplication*, in *Proceedings of the 30th Annual IEEE Symposium on Foundations of Computer Science*, IEEE, Piscataway, NJ, 1989, pp. 332–337.
- [29] P. M. VAIDYA, *A new algorithm for minimizing convex functions over convex sets*, in *Proceedings of the 30th Annual IEEE Symposium on Foundations of Computer Science*, IEEE, Piscataway, NJ, 1989, pp. 338–343.
- [30] J. VILLAVICENCIO AND M. D. GRIGORIADIS, *Approximate Lagrangian decomposition using a modified Karmarkar logarithmic potential*, in *Network Optimization*, Lecture Notes in Economics and Mathematical Systems 450, P. Pardalos, D. Hearn, and W. Hager, eds., Springer-Verlag, Berlin, 1995, pp. 471–485.
- [31] N. YOUNG, *Sequential and parallel algorithms for mixed packing and covering*, in *Proceedings of the 42nd Annual IEEE Symposium on Foundations of Computer Science*, IEEE, Piscataway, NJ, 2001, pp. 538–546.

TYPICAL PROPERTIES OF WINNERS AND LOSERS IN DISCRETE OPTIMIZATION*

RENE BEIER[†] AND BERTHOLD VÖCKING[‡]

Abstract. We present a probabilistic analysis of a large class of combinatorial optimization problems containing all *binary optimization problems* defined by linear constraints and a linear objective function over $\{0, 1\}^n$. Our analysis is based on a semirandom input model that preserves the combinatorial structure of the underlying optimization problem by parameterizing which input numbers are of a stochastic and which are of an adversarial nature. This input model covers various probability distributions for the choice of the stochastic numbers and includes *smoothed analysis* with Gaussian and other kinds of perturbation models as a special case. In fact, we can exactly characterize the smoothed complexity of binary optimization problems in terms of their worst-case complexity: A binary optimization problem has polynomial smoothed complexity if and only if it admits a (possibly randomized) algorithm with pseudo-polynomial worst-case complexity.

Our analysis is centered around structural properties of binary optimization problems, called *winner*, *loser*, and *feasibility gap*. We show that if the coefficients of the objective function are stochastic, then the gap between the best and second best solution is likely to be of order $\Omega(1/n)$. Furthermore, we show that if the coefficients of the constraints are stochastic, then the slack of the optimal solution with respect to this constraint is typically of order $\Omega(1/n^2)$. We exploit these properties in an adaptive rounding scheme that increases the accuracy of calculation until the optimal solution is found. The strength of our techniques is illustrated by applications to various NP-hard optimization problems from mathematical programming, network design, and scheduling for which we obtain the first algorithms with polynomial smoothed/average-case complexity.

Key words. optimization problems, average-case analysis, smoothed analysis

AMS subject classification. 68Q25

DOI. 10.1137/S0097539705447268

1. Introduction. Many combinatorial optimization problems have an objective function or constraints specified in terms of real numbers representing natural quantities such as time, weight, distance, or utility. This includes some well-studied optimization problems such as traveling salesperson, shortest path, minimum spanning tree as well as various scheduling and packing problems. When analyzing the complexity of algorithms for such problems, we usually assume that these numbers are integers or rational numbers with a finite length representation. The hope is that it suffices to measure and compute with some bounded precision in order to identify an optimal or close to optimal solution. In fact, if real numbers occur only in the objective function and if this objective function is well behaved (e.g., a linear function), then calculating with reasonable approximations of the input numbers yields a feasible solution whose objective value is at least close to the optimal objective value. More problematically, however, if the constraints are defined by real numbers, then calculating with rounded input numbers might miss relevant solutions or might even produce infeasible solutions.

*Received by the editors September 27, 2004; accepted for publication (in revised form) July 23, 2005; published electronically February 21, 2006.

<http://www.siam.org/journals/sicomp/35-4/44726.html>

[†]Max-Planck-Institut für Informatik, Stuhlsatzenhausweg 85, D-66123 Saarbrücken, Germany (rbeier@mpi-inf.mpg.de). The work of this author was supported by a DAAD-Postdoctoral Fellowship.

[‡]Department of Computer Science, RWTH Aachen, D-66123, Saarbrücken, Germany (voecking@cs.rwth-aachen.de). The work of this author was supported in part by the EU within the 6th Framework Programme under contract 001907 (DELIS) and by DFG grant Vo889/2-1.

How can one solve optimization problems (efficiently) on a computer when not even the input numbers can be specified exactly? In practice, optimization problems in which real numbers occur in the input are solved by simply rounding the real numbers more or less carefully. Fortunately, this approach seems to yield reasonable results. We seek a theoretically founded explanation why this rounding approach usually works. Studying this issue under worst case assumptions does not make very much sense as, in the worst case, the smallest inaccuracy might lead to an infeasible or utterly suboptimal solution. This question needs to be studied in a stochastic model. In the following probabilistic analysis, we will show that, under some reasonable and quite general stochastic assumptions, one can usually round real-valued input numbers after only a logarithmic number of bits without changing the optimal solution. In fact, our probabilistic analysis goes far beyond the point of explaining phenomena occurring in practice. We are able to provide algorithms with polynomial average-case complexity (more precisely, polynomial smoothed complexity) for a quite general class of discrete optimization problems. Our analysis covers various well-studied NP-hard discrete optimization problems from mathematical programming, network design, and scheduling such as multidimensional knapsack, constrained spanning tree, or scheduling to minimize the weighted number of tardy jobs.

1.1. A semirandom input model for discrete optimization problems.

We consider optimization problems defined over a vector of n binary variables $x = (x_1, \dots, x_n)$. The set of feasible solutions is described by the intersection of a ground set of solutions $\mathcal{S} \subseteq \{0, 1\}^n$ and solutions that satisfy linear constraints of the form $w^T x \leq t$ or $w^T x \geq t$. The ground set \mathcal{S} of solutions can be specified arbitrarily. While the part that specifies \mathcal{S} is adversarial, we assume that the coefficients in the additional linear constraints are random or randomly perturbed real numbers. The reason for distinguishing a stochastic and an adversarial part of the input is that we do not want the randomization destroying the combinatorial structure of the underlying optimization problem. A similar choice between stochastic and adversarial applies to the objective function. If it is chosen to be adversarial, then our model covers arbitrary functions $f : \mathcal{S} \rightarrow \mathbb{R}$. Our probabilistic analysis, however, can also handle stochastic objective functions that are linear, that is, the objective is of the form *minimize* (or *maximize*) $c^T x$, where c is a vector of random or randomly perturbed real valued coefficients c_1, \dots, c_n . In the following, we use the phrase *stochastic expression* as a generic term for the linear expressions $c^T x$ and $w^T x$ occurring in the objective function and the constraints, respectively. The number of stochastic expressions is denoted by $k \geq 1$ and the number of stochastic constraints by $k' \in \{k - 1, k\}$, depending on whether or not the objective function is stochastic. For $k' \geq 1$ let \mathcal{B}_j denote the set of solutions that satisfy the j th constraint for all $j \in [k']$. The set of feasible solutions for a given problem instance is then $\mathcal{S} \cap \mathcal{B}_1 \cap \dots \cap \mathcal{B}_{k'}$.

The coefficients in the stochastic expressions are specified by independent continuous probability distributions with domain \mathbb{R} . Different coefficients might be drawn according to different distributions. The only restriction on these distributions is that their density is bounded from above. Assuming bounded densities is necessary, as otherwise worst-case instances could be approximated arbitrarily well by specifying distributions with very high density. At this point, let us remark that the density of a continuous random variable is not uniquely defined. In fact, the density function can be changed on any set of points of measure 0 without affecting the distribution function. As usual, we ignore this trifling indeterminacy; that is, when saying that the density of a variable is bounded we mean that the variable admits a bounded

density function. For a given variable, the supremum of its density function is called its *density parameter*. We will see that the maximum density parameter over the distributions of the different coefficients plays an important role in our analysis. This parameter is denoted by ϕ . Intuitively, ϕ can be seen as a measure specifying the concentration of random instances around the worst case. A worst-case instance can be interpreted as a stochastic instance in which the probability measure for each stochastic number is mapped to a single point. Thus, the larger ϕ , the closer we are to a worst-case analysis.

In our probabilistic analysis we assume that the objective function defines a unique ranking among all solutions in $\{0, 1\}^n$ according to the objective function. Observe that, if the objective function is stochastic, then the coefficients are continuous random variables. Hence, the probability that there exist solutions with the same objective function value is 0. In other words, a unique ranking is given with probability 1. Recall that the objective function does not have to be linear if it is adversarial, but if it is linear, i.e., of the form $c^T x$, $c \in \mathbb{Q}^n$, then a unique ranking can always be enforced by encoding the lexicographical order among the solutions into the less significant bits of the objective function without changing the computational complexity of the underlying optimization problem by more than a polynomial factor. In fact, most of the algorithmic problems that we will study have algorithms that implicitly realize a unique ranking. In this case, one does not even need an explicit encoding. Given a unique ranking, we aim at finding the *winner*, i.e., the highest ranked solution in $\mathcal{S} \cap \mathcal{B}_1 \cap \dots \cap \mathcal{B}_k$. In the following, optimization problems satisfying all the conditions above are called *binary optimization problems with stochastic expressions* or, for short, *binary optimization problems*.

Smoothed analysis. The framework of smoothed analysis was introduced by Spielman and Teng in [29]. They assume that first an adversary specifies all coefficients w in the constraint matrix such that the norm of each row vector is at most 1. Then these adversarial numbers are slightly perturbed by adding an independent random number drawn according to a Gaussian distribution with mean 0 and a specified standard deviation $\sigma > 0$. Spielman and Teng prove a running time for the simplex algorithm under the shadow vertex pivot rule that is polynomial in the number of variables and constraints as well as in $\frac{1}{\sigma}$. Similar results have been obtained for other problems [2, 4, 5, 7, 30]. Our probabilistic analysis is not restricted to the model of smoothed analysis, but we use this nice framework to illustrate our results.

We generalize smoothed analysis as follows. At first, we do not necessarily perturb all input numbers but only the coefficients in the stochastic expressions. Initially, an adversary chooses all input numbers. The adversarial choice for the coefficients that shall be stochastic is restricted to real numbers from $[0, 1]$ or $[-1, 1]$, depending on whether the domain should be nonnegative or also include negative numbers. Then a random perturbation slightly changes the coefficients in the stochastic constraints by adding an independent random number to each of them. These random numbers are drawn according to a specified family of probability distributions satisfying the following conditions. Let $f : \mathbb{R} \rightarrow \mathbb{R}_{\geq 0}$ be any density function such that $\sup_s (f(s)) = 1$ and $\int |s|f(s)ds$ is finite, that is, the random variable described by f has a finite expected absolute value. Function f is called the *perturbation model*. For $\phi \geq 1$, we define f_ϕ by scaling f , that is, $f_\phi(s) = \phi f(s/\phi)$, for every $s \in \mathbb{R}$. This way, the density parameter of f_ϕ is ϕ . We obtain a ϕ -*perturbation* according to the perturbation model f by adding an independent random variable with density function f_ϕ to each stochastic input number. For example, one obtains the Gaussian perturbation model from [29] by choosing f to be the Gaussian density with standard deviation $(2\pi)^{-1/2}$.

A nonnegative domain can be obtained, e.g., by choosing f to be the density of the uniform distribution over $[0, 1]$. In [29] the running time is described in terms of the standard deviation σ . In contrast, we describe the running time in terms of the density parameter ϕ . For the Gaussian and the uniform distribution these two parameters are closely related; in both cases, ϕ is proportional to $\frac{1}{\sigma}$.

Let us illustrate our semirandom input model by an example. The minimum spanning tree problem is to find a spanning tree in a given graph that has minimum weight. In the binary program formulation of this problem there is a variable x_e for each edge $e \in E$. Thus, n corresponds to the number of edges. A 0/1 solution x is feasible if the edges in the set $\{e \in E \mid x_e = 1\}$ form a spanning tree. Let \mathcal{S} denote the set of all solutions satisfying this condition. The combinatorial structure described by \mathcal{S} should not be touched by our randomization. It makes sense, however, to assume that the objective function is stochastic as its coefficients describe measured quantities. So we may assume that these coefficients are perturbed with uniform ϕ -perturbations, that is, each of these coefficients correspond to the sum of an adversarial number from $[0, 1]$ and an independent random number drawn uniformly from $[0, \phi^{-1}]$. In the constrained minimum spanning tree problem [12], each edge does not only have weights but additionally a cost c_e . Now one seeks for the minimum weight spanning tree satisfying the linear cost constraint $\sum_e c_e x \leq T$. This additional constraint corresponds to a subset $\mathcal{B}_1 \subseteq \{0, 1\}^{|E|}$ so that $\mathcal{B}_1 \cap \mathcal{S}$ is the set of feasible solutions. Due to the additional constraint, the problem becomes NP-hard. We will see, however, that the problem has “polynomial smoothed complexity” assuming that either the objective function or the additional constraint is stochastic.

1.2. How accurately do we need to calculate?. More precisely, we ask how many bits of each stochastic input number do we need to reveal in order to determine the winner? We say that the winner is *determined* by revealing some number of the bits, when there is only one possible candidate for the winner regardless of the outcomes of the unrevealed bits.

THEOREM 1. *Consider any instance of a binary optimization problem Π . Let $n \geq 1$ denote the number of binary variables and $k \geq 1$ the number of stochastic expressions.*

- (a) *Suppose the expected absolute value $\mathbf{E}[|w|]$ of every stochastic coefficient w is bounded from above by $\mu > 0$. The number of bits in front of the floating point of any stochastic coefficient w is bounded by $O(\max\{1, \log(k\mu n/p)\})$, with probability at least $1 - p$, for any $0 < p < 1$.*
- (b) *Let $\phi \geq 1$ denote the maximum density parameter, that is, all density functions are upper-bounded by ϕ . The winner is determined when revealing $O(\log(k\phi n/p))$ bits after the binary point of each stochastic coefficient, with probability at least $1 - p$, for any $0 < p < 1$.*

One can always decrease ϕ or μ without changing the problem by scaling the appropriate stochastic expression (and the respective bound, where applicable) by some factor $\gamma > 0$. As the corresponding distribution has mean $\gamma\mu$ and density parameter ϕ/γ , this kind of scaling does not change the overall number of bits that need to be revealed as $\log(\mu nk) + \log(\phi nk) = \log(\gamma\mu nk) + \log(\frac{1}{\gamma}\phi nk)$. The scaling transfers only significant bits from positions before the binary points to a position after the binary point. One can eliminate one parameter by normalizing the distributions such that one parameter becomes 1. In case of a smoothed analysis, the right way to scale the input numbers is already built into the model. According to our definitions, the density function f specifying the perturbation model has to have a finite expected

absolute value. For any fixed model of perturbation, $\int |s|f(s)ds = O(1)$. In particular, the expected absolute value of the density function f_ϕ is $O(\phi^{-1})$. Taking into account that the domain of the initial adversarial choices for the stochastic coefficients is $[-1, 1]$ or $[0, 1]$, we observe that ϕ -perturbations yield coefficients with an expected absolute value of at most $\mu = O(1 + \frac{1}{\phi})$. In order to simplify the notation, our model of smoothed analysis is restricted to density parameters $\phi \geq 1$. This leads to the following result on the overall number of bits that need to be revealed per stochastic input number.

COROLLARY 2. *For any fixed perturbation model f and any density parameter $\phi > 1$, the winner is determined when revealing $O(\log(k\phi n/p))$ bits of each stochastic coefficient, with probability at least $1 - p$, for any $0 < p < 1$.*

Let us explain the concepts and ideas behind the analysis for Theorem 1. Part (a) of the theorem follows simply by applying the Markov inequality to the expected absolute values of the individual coefficients. The interesting part of the theorem is stated in (b). In order to identify the winner one needs to *isolate* the winner from other feasible solutions having a worse objective value. Furthermore, one needs to *separate* the winner from those infeasible solutions that have a better objective value than the winner. Our analysis is based on a *generalized isolating lemma*—i.e., a generalization of the well-known isolating lemma by Mulmuley, Vazirani, and Vazirani [22]—and a novel *separating lemma*.

The isolating lemma was originally presented in an article about RNC algorithms for perfect matchings [22]. It is known, however, that the lemma does not only apply to the matching problem but to general binary optimization problems with a linear objective function. The lemma states that the optimal solution of a binary optimization problem is unique with a probability of at least $\frac{1}{2}$ when choosing the coefficients of the objective function independently, uniformly at random from the set $\{1, 2, \dots, 2n\}$. This is a very surprising and counterintuitive result as there might be an exponential number of feasible solutions whose objective values all fall into a polynomially large set, namely the set $\{1, 2, \dots, 2n^2\}$, so that one can expect that an exponential number of solutions are mapped to the same objective value. The reason why the winner nevertheless is isolated is that the objective values of different solutions are not independent but the solutions represent subsets over a ground set of only n random numbers, and one of these numbers will be part of the best but not the second best solution. We adapt the isolating lemma toward our continuous setting and generalize it toward continuous probability distributions with bounded density as described in section 1.1. In particular, different coefficients may follow different continuous probability distributions. Suppose only the objective function is stochastic, and the feasible region is fixed arbitrarily. Let ϕ denote the maximum density parameter over all coefficients in the objective functions. Define the *winner gap* to be the difference between the objective value of the best and the second-best feasible solution, provided there are at least two feasible solutions. The generalized isolating lemma states that, for every $\epsilon \in [0, 1]$, the winner gap is lower bounded by $\frac{\epsilon}{2\phi n}$ with probability at least $1 - \epsilon$, and this bound is tight. As a consequence, it suffices to reveal only $O(\log(\phi n))$ bits of each coefficient of the objective function in order to identify the winner, with high probability.

We accompany the isolating lemma with a novel separating lemma, enabling us to separate the winner from infeasible solutions with better objective value than the winner. For the time being, consider any binary optimization problem in which a single constraint is stochastic. The difficulty in checking the feasibility with respect to this constraint is that it might be likely that there are many solutions that are

exponentially close to the constraint hyperplane. Nevertheless, we will see that the optimal solution can be identified by inspecting only a logarithmic number of bits per input number, with high probability. The reason is that we do not need to check the feasibility of all solutions but only of some particular solutions. The *losers* are those solutions that have a rank higher than the winner, but they are infeasible because of the stochastic constraint. The *loser gap* is defined to be the minimal amount by which a loser (except for the solution 0^n) exceeds the constraint threshold. The separating lemma shows that, for every $\epsilon \in [0, 1]$, the loser gap is larger than $\frac{\epsilon}{\phi n^2}$ with a probability of at least $1 - \epsilon$. Let us try to give some intuition about this result. If there are only a few losers, then one can imagine that none of them comes very close to a random or randomly perturbed hyperplane. However, there might be an exponential number of losers. In this case, however, the winner has a relatively low rank as there is an exponential number of solutions better than the winner; but this is very unlikely if the constraint hyperplane is likely to come very close to the good solutions which correspond to the losers. Seeing it the other way around, if there are many losers, then the hyperplane is likely to be relatively far away from the losers, which might intuitively explain the phenomenon described by the separating lemma. Besides the loser gap, we study the so-called *feasibility gap* corresponding to the slack of the optimal solution with respect to the stochastic constraint. We can show that the lower bound for the loser gap holds also for the feasibility gap. In fact, our analysis for loser and feasibility gaps is heavily based on symmetry properties between them.

When analyzing the winner gap, we assume that the objective function is random and the feasible region is fixed. In contrast, when we study the loser and feasibility gaps, we assume that the feasible region is determined by random constraints and that the objective function is fixed. In other words, the random expressions defining the objective function and the constraints are assumed to be stochastically independent. In fact, if the feasible region and the objective function are correlated, then winner, loser, and feasibility cannot be lower bounded by the reciprocal of a polynomial. The optimization variant of the subset-sum problem (i.e., knapsack with profits equal to weights) is a simple counterexample. Lueker [19] proved that random instances of the subset-sum problem have usually exponentially small gaps.

2. Analysis of the gap properties. In this section, we formally define the winner, loser, and feasibility gap and prove lower bounds for their size. We begin with an analysis of the winner gap. Afterwards we study the loser and the feasibility gap; first for a single constraint and then for multiple constraints. Finally, we apply these results to prove Theorem 1.

2.1. The winner gap. We consider an instance of a discrete optimization problem whose solutions are described by n binary variables x_1, \dots, x_n . The set of feasible solutions (ground set intersected with the constraints) is now denoted by $\mathcal{S} \subseteq \{0, 1\}^n$. Fix some arbitrary set \mathcal{S} with at least two solutions. The objective function is denoted by $c^T x$. The numbers $c_i \in \mathbb{R}$, $i = 1, \dots, n$, are assumed to be stochastic, that is, they are treated as independent random variables following possibly different continuous probability distributions with bounded density. Without loss of generality, we consider a maximization problem. Let $x^* = \operatorname{argmax}\{c^T x \mid x \in \mathcal{S}\}$ denote the winner and $x^{**} = \operatorname{argmax}\{c^T x \mid x \in \mathcal{S} \setminus \{x^*\}\}$ the second best solution. The *winner gap* Δ is defined to be the difference between the objective values of a best and second best solution, that is, $\Delta = c^T x^* - c^T x^{**}$.

LEMMA 3 (generalized isolating lemma). *Fix the feasible region \mathcal{S} . Let ϕ_i denote the density parameter of c_i , for $1 \leq i \leq n$, and $\phi = \max_i \phi_i$. For every $\epsilon \geq 0$,*

$\Pr [\Delta < \epsilon] \leq 2\epsilon \sum_{i \in [n]} \phi_i \leq 2\epsilon \phi n.$

Proof. At first we observe, if there is a variable x_i that takes the same value in all feasible solutions, then this variable does not affect the winner gap and it can be ignored. Thus, without loss of generality, for every $i \in [n]$, there are at least two feasible solutions whose vectors differ in the i th bit, i.e., with respect to the i th variable. Under this assumption, we can define the winner gap with respect to bit position $i \in [n]$ by

$$(1) \quad \Delta_i = c^T x^* - c^T y,$$

where $x^* = \operatorname{argmax}\{c^T x \mid x \in \mathcal{S}\}$, $y = \operatorname{argmax}\{c^T x \mid x \in \mathcal{S}, x_i \neq x_i^*\}$. In other words, Δ_i is the difference between the objective value of the winner x^* and the value of a solution y that is best among those solutions that differ in the i th bit from x^* , i.e., the best solution in $\{x \in \mathcal{S} \mid x_i \neq x_i^*\}$.

Clearly, the best solution, $x^* = (x_1^*, \dots, x_n^*)$, and the second best solution, $x^{**} = (x_1^{**}, \dots, x_n^{**})$, differ in at least one bit, that is, there exists $i \in [n]$ such that $x_i^* \neq x_i^{**}$. If the best and second best solution differ in the i th bit, then $\Delta = \Delta_i$. Thus, Δ is guaranteed to take a value also taken by at least one of the variables $\Delta_1, \dots, \Delta_n$. In the following, we will prove $\Pr [\Delta_i < \epsilon] \leq 2\epsilon \phi_i$, for $1 \leq i \leq n$. Thus,

$$\Pr [\Delta < \epsilon] \leq \Pr [\exists i \in [n] : \Delta_i < \epsilon] \leq \sum_{i \in [n]} \Pr [\Delta_i < \epsilon] \leq 2\epsilon \sum_{i \in [n]} \phi_i,$$

which implies the lemma.

Let us fix an index $i \in [n]$. It remains to show $\Pr [\Delta_i < \epsilon] \leq 2\epsilon \phi_i$. We partition \mathcal{S} , the set of feasible solutions, into two disjoint subsets $\mathcal{S}_0 = \{x \in \mathcal{S} \mid x_i = 0\}$ and $\mathcal{S}_1 = \{x \in \mathcal{S} \mid x_i = 1\}$. Now suppose all random variables $c_k, k \neq i$ are fixed arbitrarily. Under this assumption, we can identify a winner among the solutions in \mathcal{S}_0 as the objective values of the solutions in \mathcal{S}_0 do not depend on c_i . Although the objective values of the solutions in \mathcal{S}_1 are not fixed, we can nevertheless identify a winner of \mathcal{S}_1 because the unknown outcome of the random variable c_i does not affect the order among the solutions in \mathcal{S}_1 . For $j \in \{0, 1\}$, let $x^{(j)}$ denote a winner among the solutions in \mathcal{S}_j . We observe $\Delta_i = |c^T x^{(1)} - c^T x^{(0)}|$ because the solutions x^* and y as defined in (1) cannot be contained in the same set $\mathcal{S}_j, j \in \{0, 1\}$. Consider the random variable $Z = c^T x^{(1)} - c^T x^{(0)}$. We have shown $\Delta_i = |Z|$. Observe that the random variable c_i appears as a simple additive term in Z . Hence, the density function of Z is a shifted variant of the density function of c_i . In particular, ϕ_i is an upper bound for the density of Z , as it is for the density of c_i . Hence, $\Pr [\Delta_i < \epsilon] = \Pr [Z \in (-\epsilon, \epsilon)] \leq 2\epsilon \phi_i$. This completes the proof of the generalized isolating lemma. \square

Next we show that the given bound in the generalized isolating lemma is tight.

LEMMA 4. *Suppose $\mathcal{S} = \{0, 1\}^n$. Let $\phi_1, \dots, \phi_n > 0$ denote an arbitrary collection of density parameters. Fix any $\alpha < \sum_{i \in [n]} \phi_i$. There is a way to define continuous probability distributions for the coefficients c_1, \dots, c_n with density parameters ϕ_1, \dots, ϕ_n , respectively, such that there exists $\epsilon > 0$ with $\Pr [\Delta < \epsilon] > 2\alpha\epsilon$.*

Proof. As $\mathcal{S} = \{0, 1\}^n$, the optimal solution x^* satisfies $x_i^* = 1 \Leftrightarrow c_i > 0$. Let $k = \operatorname{argmin}_i(|c_i|)$. The second best solution x^{**} satisfies $x_i^{**} = x_i^*$, for all $i \neq k$, and $x_k^{**} = 1 - x_k^*$. Thus, $\Delta = \min_i(|c_i|)$. Assume the density functions of c_1, \dots, c_n are continuous and take their maximum value ϕ_i at 0. We study the density of the random variable Δ at 0,

$$\lim_{\substack{\epsilon \rightarrow 0 \\ \epsilon > 0}} \frac{\Pr [\Delta < \epsilon]}{\epsilon} = \lim_{\substack{\epsilon \rightarrow 0 \\ \epsilon > 0}} \frac{\Pr [\exists i \in [n] : |c_i| < \epsilon]}{\epsilon} = \sum_{i \in [n]} \lim_{\substack{\epsilon \rightarrow 0 \\ \epsilon > 0}} \frac{\Pr [|c_i| < \epsilon]}{\epsilon},$$

where the last equality is due to the fact that for any $S \subset \mathcal{S}$ with $|S| > 1$,

$$\lim_{\substack{\epsilon \rightarrow 0 \\ \epsilon > 0}} \frac{\Pr [\bigwedge_{i \in S} |c_i| < \epsilon]}{\epsilon} = 0.$$

Now

$$\lim_{\substack{\epsilon \rightarrow 0 \\ \epsilon > 0}} \frac{\Pr [|c_i| < \epsilon]}{\epsilon} = \lim_{\substack{\epsilon \rightarrow 0 \\ \epsilon > 0}} \frac{\Pr [c_i \in (-\epsilon, \epsilon)]}{\epsilon} = 2\phi_i.$$

As a consequence,

$$\lim_{\substack{\epsilon \rightarrow 0 \\ \epsilon > 0}} \frac{\Pr [\Delta < \epsilon]}{\epsilon} = 2 \sum_{i \in [n]} \phi_i,$$

so that, for any $\alpha < \sum_{i \in [n]} \phi_i$, there exists $\epsilon > 0$ with $\Pr [\Delta < \epsilon] > 2\alpha\epsilon$. \square

2.2. Loser and feasibility gaps for a single constraint. We consider an instance of an optimization problem over n binary variables. The objective function can be fixed arbitrarily; we rank all solutions (feasible and infeasible) according to their objective value in nonimproving order. Solutions with the same objective values are ranked in an arbitrary but fixed fashion. The feasible region is described by a subset $\mathcal{S} \subseteq \{0, 1\}^n$ intersected with the half space \mathcal{B} described by an additional linear constraint. Without loss of generality, the constraint is of the form $w^T x \leq t$. The set \mathcal{S} and the threshold t are assumed to be fixed. The coefficients w_1, \dots, w_n correspond to independent random variables following possibly different continuous probability distributions with bounded density. The *winner*, denoted by x^* , is the solution with highest rank in $\mathcal{S} \cap \mathcal{B}$. The *feasibility gap* is defined by

$$\Gamma = \begin{cases} t - w^T x^* & \text{if } \mathcal{S} \cap \mathcal{B} \neq \emptyset, \text{ and} \\ \perp & \text{otherwise.} \end{cases}$$

In other words, Γ corresponds to the slack of the winner with respect to the constraint $w^T x \leq t$. Observe that x^* might be undefined if there is no feasible solution. In this case, the random variable Γ takes the value \perp (*undefined*).

A solution in \mathcal{S} is called a *loser* if it has a higher rank than x^* , that is, the losers are those solutions from \mathcal{S} that have a better rank than the winner, but they are cut off by the constraint $w^T x \leq t$. The set of losers is denoted by \mathcal{L} . If there is no winner, as there is no feasible solution, then we define $\mathcal{L} = \mathcal{S}$. The *loser gap* is defined by

$$\Lambda = \begin{cases} \min\{w^T x - t \mid x \in \mathcal{L}\} & \text{if } \mathcal{L} \neq \emptyset, \text{ and} \\ \perp & \text{otherwise.} \end{cases}$$

In case $\mathcal{L} \neq \emptyset$, the loser that determines Λ , i.e., $\operatorname{argmin}_{x \in \mathcal{L}} \{w^T x\}$, is called *minimal loser*. Figure 1 illustrates these definitions.

Our goal is to lower bound Γ and Λ . Observe that the solution 0^n is different from all other solutions in \mathcal{S} , as its feasibility does not depend on the outcome of the random coefficients w_1, \dots, w_n . Suppose $0^n \in \mathcal{S}$ and 0^n has the highest rank among all solutions in \mathcal{S} . Then one can enforce $\Gamma = 0$ by setting $t = 0$. Similarly, one can enforce $\Lambda \rightarrow 0$ for $t \rightarrow 0$. For this reason, we need to exclude the solution 0^n from our analysis.

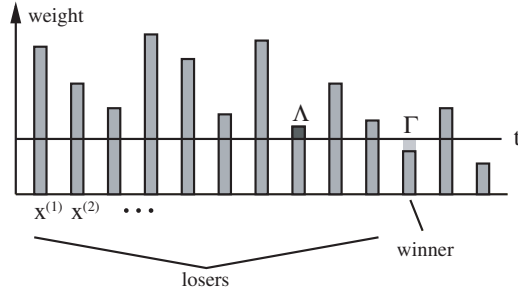


FIG. 1. Loser and feasibility gap: Solutions are listed from left to right according to the ranking. The height of the bars correspond to the weight $w^T x$. The winner is the first solution whose weight satisfies the threshold t . The feasibility gap Γ is the slack of the winner with respect to the constraint $w^T x \leq t$. All solutions left of the winner are losers. The loser gap Λ is the smallest amount by which any loser violates the threshold t .

LEMMA 5 (separating lemma). Let ϕ_i denote the density parameter of w_i , for all $i \in [n]$, and define $\phi = \max_{i \in [n]} \phi_i$. Suppose $0^n \notin \mathcal{S}$. For every $\epsilon \geq 0$, $\Pr[\Gamma < \epsilon] \leq \epsilon n \sum_{i \in [n]} \phi_i \leq \epsilon \phi n^2$ and $\Pr[\Lambda < \epsilon] \leq \epsilon n \sum_{i \in [n]} \phi_i \leq \epsilon \phi n^2$.

Proof. We will heavily use symmetry properties between the two gaps. First, we will prove $\Pr[\Lambda < \epsilon] \leq \epsilon \sum_{i \in [n]} \phi_i$ under the assumption that the ranking satisfies a certain monotonicity property. Next, we will show that any probabilistic lower bound of this kind that holds for the loser gap with respect to any given threshold $t \in \mathbb{R}$ transfers to the feasibility gap and vice versa. Because of this symmetry property, the probability that the loser gap is smaller than ϵ is also at most $\epsilon \sum_{i \in [n]} \phi_i$ under the same monotonicity assumption. Afterwards, we will show that the monotonicity assumption for the feasibility gap can be dropped at the cost of an extra factor n , thereby achieving an upper bound of $\epsilon n \sum_{i \in [n]} \phi_i$ on the probability that the feasibility gap is smaller than ϵ . Finally, by applying the symmetry argument in the other direction, we obtain the same result for the loser gap.

A ranking of the solutions is called *monotone* if all pairs of solutions $x, y \in \mathcal{S}$, x having a higher rank than y , satisfy that there exists $i \in [n]$ with $x_i > y_i$. When considering the binary solution vector as subsets of $[n]$, a ranking is *monotone* if each subset S is ranked higher than all its proper subsets $T \subset S$. This property is naturally satisfied for maximization problems having a linear objective function with positive coefficients, but also if all solutions in \mathcal{S} have the same number of ones.

LEMMA 6. Suppose $0^n \notin \mathcal{S}$ and the ranking is monotone. Then $\Pr[\Lambda < \epsilon] \leq \epsilon \sum_{i \in [n]} \phi_i$.

Proof. Fix $t \in \mathbb{R}$ arbitrarily. As in the proof for the winner gap, we define n random variables $\Lambda_1, \dots, \Lambda_n$ with maximum densities ϕ_1, \dots, ϕ_n such that at least one of them takes the value of Λ . For $i \in [n]$, define $\mathcal{S}_i = \{x \in \mathcal{S} \mid x_i = 1\}$ and $\bar{\mathcal{S}}_i = \mathcal{S} \setminus \mathcal{S}_i$. Let $\bar{x}^{(i)}$ denote the winner from $\bar{\mathcal{S}}_i$, i.e., the solution with the highest rank in $\bar{\mathcal{S}}_i \cap \mathcal{B}$. Now let \mathcal{L}_i denote the set of losers from \mathcal{S}_i when assuming that $\bar{x}^{(i)}$ is the winner, that is, $\mathcal{L}_i = \{x \in \mathcal{S}_i \mid x \text{ has a higher rank than } \bar{x}^{(i)}\}$. If $\bar{x}^{(i)}$ does not exist, then we set $\mathcal{L}_i = \mathcal{S}_i$. Now define the minimal loser of \mathcal{L}_i , $x_{\min}^{(i)} = \operatorname{argmin}\{w^T x \mid x \in \mathcal{L}_i\}$, and

$$\Lambda_i = \begin{cases} w^T x_{\min}^{(i)} - t & \text{if } \mathcal{L}_i \neq \emptyset, \text{ and} \\ \perp & \text{otherwise.} \end{cases}$$

Observe that \mathcal{L}_i is not necessarily a subset of \mathcal{L} , as $\bar{x}^{(i)}$ can have a lower rank than x^* .

In fact, $x_{\min}^{(i)}$ can be feasible so that Λ_i can take negative values. The reason for this “wasteful” definition is that it yields some kind of independence that we will exploit in the following arguments.

Claim A. $\Pr[\Lambda_i \in [0, \epsilon)] \leq \epsilon\phi_i$. This claim can be seen as follows. The definitions above ensure $\mathcal{L}_i \subseteq \mathcal{S}_i$ while $\bar{x}^{(i)} \in \bar{\mathcal{S}}_i$. Suppose all variables $w_j, j \neq i$ are fixed arbitrarily. The winner $\bar{x}^{(i)}$ can be determined without knowing the outcome of w_i as $\bar{x}^{(i)} \in \bar{\mathcal{S}}_i$ and for all solutions in $\bar{\mathcal{S}}_i$ the i th entry is zero. Observe that \mathcal{L}_i is fixed as soon as $\bar{x}^{(i)}$ is fixed, and so is $x_{\min}^{(i)}$, as the i th bit of all losers in \mathcal{L}_i is one. Hence, w_i is not affected by fixing $x_{\min}^{(i)}$. As the i th bit of $x_{\min}^{(i)}$ is one, the random variable Λ_i can be rewritten as $\Lambda_i = w^T x_{\min}^{(i)} - t = \kappa + w_i$, where κ depends on only the fixed coefficients $w_j, j \neq i$, and w_i is a random variable with density at most ϕ_i . Consequently, the density of Λ_i is bounded from above by ϕ_i . (The same is true if $\bar{x}^{(i)}$ does not exist so that $\mathcal{L}_i = \mathcal{S}_i$.) This upper bound on the density of Λ_i immediately implies $\Pr[\Lambda_i \in [0, \epsilon)] \leq \epsilon\phi_i$.

Claim B. If $\Lambda \neq \perp$, then there exists $i \in [n]$ such that Λ takes the value of Λ_i . To prove this claim, let us first assume that x^* exists and $\mathcal{L} \neq \emptyset$. Let $x_{\min} \in \mathcal{L}$ denote the *minimal loser*, i.e., $x_{\min} = \operatorname{argmin}\{w^T x \mid x \in \mathcal{L}\}$. By definition, x_{\min} has a higher rank than x^* . Because of the monotonicity of the ranking, there exists $i \in [n]$ such that $x^* \in \bar{\mathcal{S}}_i$ and $x_{\min} \in \mathcal{S}_i$. From $x^* \in \bar{\mathcal{S}}_i$, we conclude $x^* = \bar{x}^{(i)}$. Consequently, $x_{\min} \in \mathcal{L} \cap \mathcal{S}_i = \mathcal{L}_i$ so that $x_{\min} = x_{\min}^{(i)}$. Hence, $\Lambda = \Lambda_i$. Now suppose x^* does not exist. Then $\mathcal{L} = \mathcal{S}$ and $\mathcal{L}_i = \mathcal{S}_i$, for all $i \in [n]$. Thus, there exists $i \in [n]$ with $x_{\min} = x_{\min}^{(i)}$ because $\mathcal{S} = \bigcup_{i \in [n]} \mathcal{S}_i$ as $0^n \notin \mathcal{S}$. Finally, if $\mathcal{L} = \emptyset$, then the claim follows immediately as $\Lambda = \perp$.

Combining the claims with the union bound gives

$$\Pr[\Lambda < \epsilon] \leq \Pr[\exists i \in [n] : \Lambda_i \in [0, \epsilon)] \leq \sum_{i \in [n]} \Pr[\Lambda_i < \epsilon] \leq \sum_{i \in [n]} \epsilon\phi_i. \quad \square$$

Next we present a symmetry property that allows the transfer of lower bounds on the loser gap to the feasibility gap and vice versa. Observe that the lower bound above holds for any given threshold $t \in \mathbb{R}$. This is important for the application of our symmetry argument. In the following, let $\Lambda(t)$ and $\Gamma(t)$ denote loser and feasibility gap with respect to threshold t , respectively.

LEMMA 7. *Suppose $0^n \notin \mathcal{S}$. For any $t \in \mathbb{R}$ and $\epsilon \geq 0$,*

$$\Pr[\Gamma(t) < \epsilon] = \Pr[\Lambda(t - \epsilon) < \epsilon].$$

Proof. We take an alternative view on the given optimization problem. We interpret the problem as a bicriteria problem. The feasible region is then the whole set \mathcal{S} . The first criterion is the rank which is to be minimized (high ranks are small numbers). The second criterion is the weight, defined by the linear function $w^T x$, which is to be minimized as well. A solution $x \in \mathcal{S}$ is called *Pareto-optimal* if there is no higher ranked solution $y \in \mathcal{S}$ with smaller (or equal) weight. For simplicity assume that no two solutions have the same weight. This assumption is justified as the probability that there are two solutions with the same weight is 0.

Next we show that winners and minimal losers of the original optimization problem correspond to Pareto-optimal solutions of the bicriteria problem. First, let us observe that the winner x^* with respect to any given weight threshold t is a Pareto-optimal solution for the bicriteria problem because there is no other solution with

a higher rank and weight at most $t \geq w^T x^*$. Moreover, for every Pareto-optimal solution x there is also a threshold t such that x is a winner, i.e., $t = w^T x$.

The same kind of characterization holds for minimal losers as well. Recall, for a given threshold t , the minimal loser is defined to be $x_{\min} = \operatorname{argmin}\{w^T x \mid x \in \mathcal{L}\}$. We claim that there is no other solution $y \in \mathcal{S}$ that simultaneously achieves a higher rank and smaller weight than x_{\min} . This can be seen as follows. Suppose $y \in \mathcal{S}$ is a solution with higher rank than x_{\min} . If $w^T y \leq t$, then $y \in \mathcal{B}$ and, hence, x_{\min} would not be a loser. However, if $w^T y \in (t, w^T x_{\min})$, then y and x_{\min} would both be losers, but y instead of x_{\min} would be minimal. Furthermore, for every Pareto-optimal solution x there is also a threshold t such that x is a loser. This threshold can be obtained by setting $t \rightarrow w^T x$, $t < w^T x$.

Now let us describe winner and loser gap in terms of Pareto-optimal solutions. Let $\mathcal{P} \subseteq \mathcal{S}$ denote the set of Pareto-optimal solutions with respect to the fixed ranking and the random weight function $w^T x$. Then feasibility and loser gaps for any given threshold $t \in \mathbb{R}$ satisfy

$$\begin{aligned} \Gamma(t) &= \min\{t - w^T x \mid x \in \mathcal{P}, w^T x \leq t\}, \\ \Lambda(t) &= \min\{w^T x - t \mid x \in \mathcal{P}, w^T x > t\}. \end{aligned}$$

For a better intuition, we can imagine that all Pareto-optimal solutions are mapped onto a horizontal line such that $x \in \mathcal{P}$ is mapped to the point $w^T x$. Then $\Gamma(t)$ is the distance from the point t on this line to the closest Pareto point *left* of t (i.e., less than or equal to t), and $\Lambda(t)$ is the distance from t to the closest Pareto point *strictly right* of t (i.e., larger than t). As a consequence, $\Gamma(t) < \epsilon$ if and only if $\Lambda(t - \epsilon) \leq \epsilon$ and, hence,

$$\Pr[\Gamma(t) < \epsilon] = \Pr[\Lambda(t - \epsilon) \leq \epsilon] = \Pr[\Lambda(t - \epsilon) < \epsilon]. \quad \square$$

Combining Lemmas 6 and 7 yields that the probability that the feasibility gap is smaller than ϵ is at most $\epsilon \sum_{i \in [n]} \phi_i$ as well, provided that the ranking is monotone and $0^n \notin \mathcal{S}$. Next we extend this result toward general rankings by breaking the original problem into suitable subproblems. We partition \mathcal{S} into the sets $\mathcal{S}^{(k)} = \{x \in \mathcal{S} \mid \sum_i x_i = k\}$, for $1 \leq k \leq n$. Observe that each of these sets contains only solutions with the same number of ones, and hence, satisfies the monotonicity condition. Let $\Gamma^{(k)}(t)$ denote the feasibility gap over the set $\mathcal{S}^{(k)}$. By our arguments above, $\Pr[\Gamma^{(k)}(t) < \epsilon] \leq \epsilon \sum_{i \in [n]} \phi_i$. Furthermore, $\Gamma(t)$ takes the value of one of the variables $\Gamma^{(k)}(t)$, $1 \leq k \leq n$, because the winner of one of the subproblems is the winner of the original problem. Applying the union bound gives $\Pr[\Gamma(t) < \epsilon] \leq \epsilon n \sum_{i \in [n]} \phi_i$. Let us remark that such a kind of argument cannot be applied directly to the loser gap. By applying Lemma 7, however, the bound for the feasibility gap holds for the loser gap as well. This completes the proof of the separating lemma. \square

2.3. Loser and feasibility gap for multiple constraints. Assume there are $k \geq 2$ stochastic constraints. Without loss of generality, these constraints are of the form $w_j^T x \leq t_j$, for $j \in [k]$, and the sets of solutions satisfying these constraints are $\mathcal{B}_1, \dots, \mathcal{B}_k$, respectively. We generalize the definition of feasibility and loser gap as follows. Given a set of solutions $\mathcal{S} \subseteq \{0, 1\}^n$ and a ranking, the *winner* x^* is the highest ranked solution in $\mathcal{S} \cap \mathcal{B}_1 \cap \dots \cap \mathcal{B}_k$. The *feasibility gap for multiple constraints* is the minimal slack of x^* over all stochastic constraints, that is, $\Gamma = \min_{j \in [k]} \{t_j - w_j^T x^*\}$, if x^* exists, and $\Gamma = \perp$, otherwise.

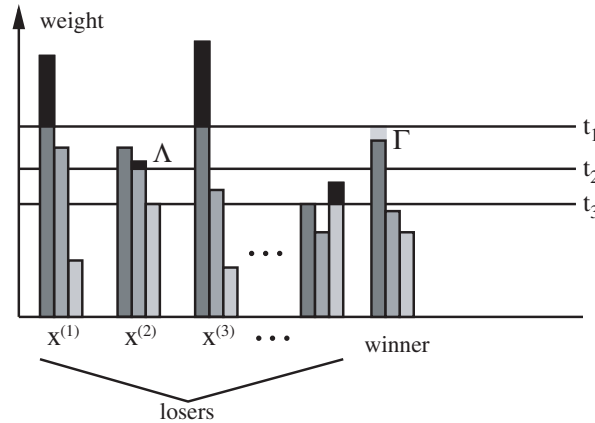


FIG. 2. Loser and feasibility gap for three constraints: Solutions are listed from left to right according to the ranking. For each solution three bars are shown representing three different weight functions. The thresholds t_1, t_2 , and t_3 apply to the first, second, and third weight function (bar), respectively. The winner is the first solution satisfying all three thresholds. The feasibility gap Γ is the smallest slack of the winner over the different constraints. All solutions left of the winner are losers. The loser gap for an individual loser is the largest amount by which this loser violates a threshold (given in black). The loser gap Λ is the minimum of the individual loser gaps over all losers.

A solution in \mathcal{L} is called a loser if it has a higher rank than x^* . Observe that a loser needs only to be infeasible with respect to one of the k constraints. In particular, it is not true that the weight values of each loser are likely to be far away from the corresponding thresholds $t_j, j \in [k]$; not even if we consider only those constraints for which the respective loser is infeasible. Fortunately, however, we do not need such a property in the application of the loser gap. For every loser, one needs only a single constraint that renders the loser infeasible. Therefore, we define the *loser gap for k constraints* by

$$\Lambda = \begin{cases} \min_{x \in \mathcal{L}} \max_{j \in [k]} \{w_j^T x - t_j\} & \text{if } \mathcal{L} \neq \emptyset, \text{ and} \\ \perp & \text{otherwise.} \end{cases}$$

Figure 2 illustrates this definition.

LEMMA 8. Let ϕ denote the maximum density parameter of all coefficients in the stochastic constraints. Suppose $0^n \notin \mathcal{S}$. For every $\epsilon \geq 0$, $\Pr[\Gamma < \epsilon] \leq \epsilon k \phi n^2$ and $\Pr[\Lambda < \epsilon] \leq \epsilon k \phi n^2$.

Proof. First we show the bound for the feasibility gap. Let x^* denote the winner and suppose $\Gamma \leq \epsilon$, for some $\epsilon \in \mathbb{R}_{\geq 0}$. Then there exists $j \in [k]$ with $t_j - w_j^T x^* \leq \epsilon$. Thus,

$$\Pr[\Gamma \leq \epsilon] \leq \sum_{j \in [k]} \Pr[t_j - w_j^T x^* \leq \epsilon].$$

For each individual $j \in [k]$, we can apply the separating lemma assuming that the set of feasible solutions with respect to all other constraints is fixed, as the coefficients in this constraint are stochastically independent from the other constraints. This way, we obtain $\Pr[\Gamma \leq \epsilon] \leq k \cdot \epsilon \phi n^2$.

Next, we turn our attention to the loser gap. Unfortunately, we cannot generalize the bound on the loser gap from one to multiple constraints in the same way as

we generalized the feasibility gap, as the loser gap for multiple constraints does not correspond to the minimal loser gap over the individual constraints. Instead we will make use of the result for the feasibility gap established above. Assume $\Lambda \leq \epsilon$, for some $\epsilon \in \mathbb{R}_{\geq 0}$. Then there exists a loser x satisfying for all $j \in [k] : w_j^T x - t_j \leq \epsilon$. Let x_L denote the loser with this property that is ranked highest. Consider a relaxed variant Π' of the given optimization problem Π where the thresholds of all stochastic constraints are increased by ϵ , i.e., we have constraints $w_j^T x \leq t_j + \epsilon, j \in [k]$. Observe that x_L is feasible in the relaxed problem Π' and, by the definition of x_L , no higher ranked solution is feasible. Thus, x_L is the winner of Π' . Since $t_j < w_j^T x_L \leq t_j + \epsilon$ for some $j \in [k]$, the feasibility gap Γ' of the relaxed problem is smaller than ϵ . Hence, $\Lambda \leq \epsilon$ implies $\Gamma' \leq \epsilon$. Finally, applying the bound $\Pr[\Gamma' \leq \epsilon] \leq \epsilon k \phi n^2$ derived in the first part of the proof yields $\Pr[\Lambda \leq \epsilon] \leq \epsilon k \phi n^2$. \square

2.4. Proof of Theorem 1. First we prove part (a) of the theorem. The probability that the absolute value of any of the kn stochastic coefficients $w_{i,j}$, ($i \in [k], j \in [n]$), is larger than $\mu nk/p$, for any $0 < p < 1$, is

$$\Pr \left[\exists(i, j) : |w_{i,j}| > \frac{\mu nk}{p} \right] \leq \sum_{i,j} \Pr \left[|w_{i,j}| > \frac{\mu nk}{p} \right] \leq \sum_{i,j} \frac{p}{kn} = p.$$

Hence, the maximum number of bits in front of the floating point over all stochastic coefficients is $O(\max\{1, \log(\mu nk/p)\})$, with a probability of at least $1 - p$.

Next we prove part (b) of the theorem. First, suppose that the objective function is the only stochastic expression. We reveal b bits after the binary point of each coefficient c_i ($1 \leq i \leq n$). Then we know the value of each c_i up to a absolute error of 2^{-b} . We will deal with this lack of precise information in terms of rounding, that is, we will think of rounding down all c_i to the next multiple of 2^{-b} causing a “rounding error” of less than 2^{-b} for each number.

LEMMA 9. *Let ϕ denote the maximum density parameter over the coefficients c_1, \dots, c_n in the objective function. When revealing b bits after the binary point of each coefficient then the winner is determined with a probability of at least $1 - 2n^2\phi/2^b$.*

Proof. Let $\lfloor c \rfloor$ be the vector that is obtained by rounding each entry c_i of vector c down to the next multiple of 2^{-b} . Consider any two solutions $x, x' \in \mathcal{S}$. Rounding changes the difference of the objective values of x and x' by

$$|(c^T x - c^T x') - (\lfloor c \rfloor^T x - \lfloor c \rfloor^T x')| = |(c - \lfloor c \rfloor)^T (x - x')| < n2^{-b},$$

as $c_i - \lfloor c_i \rfloor < 2^{-b}$, for all $i \in [n]$. Hence, if the winner gap Δ (with respect to the exact coefficients c_1, \dots, c_n) is at least $n2^{-b}$, then the winner is sufficiently isolated and the rounding cannot affect the optimality of the winner. In this case the winner is determined by revealing only b bits of each coefficient c_i . Applying the generalized isolating lemma with $\epsilon = n2^{-b}$ we obtain $\Pr[\Delta < n2^{-b}] \leq \frac{2n^2\phi}{2^b}$ and the lemma follows. \square

Next suppose only some of the constraints are stochastic and the objective function is fixed arbitrarily. Let k' denote the number of stochastic constraints. Without loss of generality, the constraints are of the form $w_j^T x \leq t_j, j \in [k']$. Assume we reveal b bits after the binary point of each coefficient in the k' constraints. We will think of it as rounding down each coefficient to the next multiple of 2^{-b} .

LEMMA 10. *Let ϕ denote the maximum density parameter over all coefficients in the stochastic constraints. When revealing b bits after the binary point of each coefficient, the winner is determined with a probability of at least $1 - k'n^3\phi/2^b$.*

Proof. As we round down, infeasible solutions might become feasible whereas feasible solutions stay feasible. (For constraints of the form $w_j^T x \geq t_j$ one would need to round up to the next multiple of 2^{-b} .) To ensure that the winner is determined it suffices to upper bound the maximum possible error in each constraint caused by the rounding of the coefficients. If this error is smaller than the loser gap, then rounding cannot change the feasibility status of any loser, i.e., all infeasible solutions that have rank higher than the winner stay infeasible.

In order to apply the bound on the loser gap given in Lemma 8, let us first assume $0^n \notin \mathcal{S}$. The rounding error in each expression is at most $n2^{-b}$. The definition of the loser gap for multiple stochastic constraints states that for every loser x there is a constraint $j \in [k']$ such that $w_j^T x - t_j \geq \Gamma$. Therefore, if $\Gamma > n2^{-b}$, then every loser stays infeasible with respect to at least one constraint after rounding. Applying Lemma 8, the probability for this event is at least $1 - k'\phi n^3/2^b$.

Including 0^n to the set of solutions \mathcal{S} can influence our analysis in two ways. First, 0^n can be a loser and might decrease the loser gap. However, rounding the coefficients w_i does not change the feasibility of this solution and the winner is not affected by 0^n . Second, 0^n can become the winner and might thereby change the set of losers. In this case, adding 0^n to \mathcal{S} can only remove solutions from \mathcal{L} and, hence, can only increase the loser gap. Thus, the above bound holds even if $0^n \in \mathcal{S}$. \square

Now suppose some constraints as well as the objective function are stochastic. We can combine the results for the winner gap and the feasibility gap as the coefficients of the objective function and the random constraints are assumed to be independent. Let $k = k' + 1$ denote the number of stochastic expressions. The probability that the winner and loser gap are sufficiently large, as described in the proofs of the two lemmas above, is $1 - k'\phi n^3/2^b - 2\phi n^2/2^b \geq 1 - k\phi n^3/2^b$, for $n \geq 2$.

This implies that the winner is determined when revealing $b = O(\log(k\phi n/p))$ bits, with a probability of at least $1 - p$, for any $0 < p < 1$. This completes the proof of Theorem 1. \square

3. Characterizing polynomial smoothed complexity. Based on the gap properties, we aim at characterizing which discrete optimization problems have polynomial time algorithms under random perturbations. We formalize this as follows. Fix any binary optimization problem Π and any perturbation model f . Let \mathcal{I}_N denote the set of all unperturbed instances of length N that an adversary may specify. The definition of the input length N needs some clarification as the coefficients in the stochastic expressions are assumed to be real numbers. We define that each of these numbers has a virtual length of one. (This way, we ensure $N \geq kn$.) The bits of the stochastic numbers can be accessed by asking an oracle in time $O(1)$ per bit. The bits after the binary point of each coefficient are revealed one by one from left to right. The deterministic part of the input can be encoded in an arbitrary fashion. For an instance $I \in \mathcal{I}_N$, let $I + f_\phi$ denote the random instance that is obtained by a ϕ -perturbation of I . We say that Π has *smoothed polynomial complexity* if and only if it admits an algorithm \mathcal{A} whose running time T satisfies

$$\exists \alpha, \beta > 0 : \text{for all } \phi \geq 1 : \text{for all } N \in \mathbb{N} : \max_{I \in \mathcal{I}_N} \mathbf{E}[(T(I + f_\phi))^\alpha] \leq \beta \phi N.$$

This definition of polynomial smoothed complexity follows more or less the way in which polynomial complexity is defined in average-case complexity theory, adding the requirement that the running time should be polynomially bounded not only in N but also in ϕ . It is not difficult to show that the assumption on the running time of \mathcal{A} is

equivalent to requiring that there exists a polynomial $P(N, \phi, \frac{1}{\epsilon})$ such that for every $N \in \mathbb{N}, \phi \geq 1, \epsilon \in [0, 1]$, the probability that the running time of \mathcal{A} exceeds $P(N, \phi, \frac{1}{\epsilon})$ is at most ϵ . Observe that this does not imply that the expected running time is polynomially bounded. To enforce expected polynomial running time, the exponent α in the definition of polynomial smoothed complexity should have been placed outside instead of inside the expectation. The reason for not defining polynomial smoothed complexity based on the expected running time is that this is not a sufficiently robust notion. For example, an algorithm with expected polynomial running time on one machine model might have expected exponential running time on another machine model. In contrast, the above definition yields a notion of polynomial smoothed complexity that does not vary among classes of machines admitting polynomial time simulations among each other. Although polynomial smoothed complexity does not always imply polynomial bounds on the expected running time, we will show that several of our algorithmic results yield expected polynomial running time on a random access machine (RAM).

We show that the smoothed complexity of a binary optimization problem Π can be characterized in terms of the worst-case complexity of Π . Theorem 1 shows that one usually needs to reveal only a logarithmic number of bits per real-valued input number. This suggests a connection between pseudo-polynomial worst-case running time and polynomial average-case complexity. For a binary optimization problem Π , let Π_u denote the corresponding optimization problem in which all numbers in the stochastic expression are assumed to be integers in unary representation instead of randomly chosen real-valued numbers. The following theorem holds for any fixed perturbation model f .

THEOREM 11. *A binary optimization problem Π has polynomial smoothed complexity if and only if $\Pi_u \in \text{ZPP}$.*

In other words, Π has polynomial smoothed complexity if it admits a (possibly randomized) algorithm whose (expected) running time for all instances is pseudo-polynomial in the stochastic constraints and polynomial in the remaining input. Notice that the expectation is over the randomization of the algorithm, not over the instances. This characterization immediately shows that strongly NP-hard optimization problems do not have polynomial smoothed complexity, unless $\text{ZPP} = \text{NP}$. This observation might not sound very surprising, as the hardness of strongly NP-hard problems does not rely on large or precisely specified input numbers. Observe, however, that the strong NP-hardness of a problem does not immediately rule out the possibility of a polynomial average-case complexity. For example, the TSP problem (on a complete graph) with edge lengths drawn uniformly at random from $[0, 1]$ might have a polynomial average-case complexity. Our theorem, however, shows that it does not have a polynomial smoothed complexity, unless $\text{ZPP} = \text{NP}$. The more sophisticated part of the theorem is the other direction stating that every binary optimization problem admitting a pseudo-polynomial time algorithm has polynomial smoothed complexity. This result is based on the generalized isolating lemma and the separating lemma. The idea is as follows. We design efficient verifiers checking whether a solution computed with a certain precision is actually the optimal solution of Π . The success probability of these verifiers is analyzed with the help of the gap properties. Using an adaptive rounding procedure, we increase the precision until we can certify that the computed solution is optimal. The overall running time of this meta-algorithm is polynomial if the algorithm computing solutions with bounded precision has pseudo-polynomial running time.

Proof. At first, we prove that the existence of a randomized pseudo-polynomial time algorithm for a binary optimization problem Π implies polynomial smoothed complexity for Π . We design an algorithm with polynomial smoothed complexity calling the pseudo-polynomial algorithm with higher and higher precision until the solution found is certified to be optimal. We describe verifiers that, based on the first b bits after the binary point of each coefficient in the stochastic expressions, either certify optimality or report FAILURE, stating that they have not sufficient information to ensure optimality. So the algorithm has access to the first b bits only, which corresponds to rounding down the numbers to the next multiple of 2^{-b} . Again we will interpret the lack of precise information as a rounding error. In the following, $\lfloor w \rfloor$ denotes the value of w rounded down to the next multiple of 2^{-b} .

Certifying optimality. For a moment assume that only the objective function is stochastic. Without loss of generality, consider a maximization problem with an objective function $c^T x$. At first, we compute an optimal solution x' for the problem with the rounded coefficients $\lfloor c_1 \rfloor, \dots, \lfloor c_n \rfloor$. To check whether x' is optimal with respect to the original cost vector c , we generate another vector \bar{c} of rounded coefficients. This time the rounding depends on the computed solution x' . For all i with $x'_i = 1$, we set $\bar{c}_i := \lfloor c_i \rfloor$ and for all i with $x'_i = 0$, we set $\bar{c}_i := \lceil c_i \rceil = \lfloor c_i \rfloor + 2^{-b}$. Observe that the function $\delta(x) = c^T x - \bar{c}^T x$ is maximal for $x = x'$. Next we compute an optimal solution x'' for the problem with the vector \bar{c} . If $x' = x''$, then x' simultaneously maximizes $\delta(x)$ and $\bar{c}^T x$. Consequently, it maximizes $\bar{c}^T x + \delta(x) = c^T x$ as well and, hence, x' corresponds to the true winner x^* . Thus, the algorithm outputs x' as a certified winner if $x' = x''$ and reports FAILURE otherwise. If the winner gap is large enough so that the winner is determined in the sense of Lemma 9, then the algorithm will always compute a certified winner. Hence, the probability that the algorithm is successful is at least $1 - 2n^2\phi/2^b$, corresponding to the bound given in Lemma 9. Observe that $\Gamma \geq n2^{-b}$ is a sufficient but not a necessary condition to certify the optimality of the winner with b revealed bits per coefficient. So the verifier might verify optimality of the computed solution even if $\Gamma < n2^{-b}$.

Certifying feasibility. Now we show how to deal with stochastic constraints. Without loss of generality, we assume that all stochastic constraints are of the form $w_j^T x \leq t_j$, $1 \leq j \leq k'$. For constraints of the form $w^T x \geq t$, we would need to round up instead of round down. First, we compute a certified winner, denoted by x' , using the rounded coefficients in the stochastic constraints. If the objective function is not stochastic, then there is no need to certify the winner. As we round down all coefficients in the stochastic constraints, we ensure that feasible solutions stay feasible. However, we have to detect infeasible solutions that become feasible due to the rounding and displace the true winner. Hence, we need to check whether x' is indeed feasible with respect to the original constraints. This would be trivial if the exact values of all constraint vectors w_1, \dots, w_k were available. However, we want to check the feasibility using only the knowledge of the first b bits after the binary point of each coefficient. Assume the solution x is infeasible with respect to the j th constraint and becomes feasible due to rounding. Then $\lfloor w_j \rfloor^T x \leq t_j < w_j^T x$ and hence $t_j - \lfloor w_j \rfloor^T x < w_j^T x - \lfloor w_j \rfloor^T x \leq n2^{-b}$, i.e., the slack of x in the j th constraint is less than $n2^{-b}$ with respect to the rounded vector $\lfloor w_j \rfloor$. Our verifier will use this property and classifies x' as possibly infeasible if it has slack less than $n2^{-b}$ for any of the k constraints with respect to the rounded coefficients. This way, we have two failure sources. First, there might be a loser that becomes feasible because of rounding. As seen in the proof of Lemma 10, this can happen only if the loser gap is smaller than $n2^{-b}$. Second, the true winner can be rejected since its slack is less than $n2^{-b}$.

This happens only if the feasibility gap is smaller than $n2^{-b}$. Due to Lemma 8, the probability that one of these events happens is at most $2k'\phi n^3/2^b$.

Adaptive rounding procedure. Consider a binary optimization problem Π with n binary variables and k stochastic expressions. Assume there exists an algorithm \mathcal{A} for the special variant of Π , where the domain of the coefficients in the stochastic expressions is restricted to \mathbb{Z} . Furthermore, assume that the worst-case running time of \mathcal{A} is bounded by some polynomial in W, n, k , and N , where W denotes the largest absolute value of any coefficient in the stochastic expressions. Recall that N specifies the size of the deterministic part of the input. We use the following adaptive rounding scheme. We start by revealing $b = \log(k\phi n^3)$ bits of each coefficient in the stochastic expressions. We obtain integer coefficients by scaling these numbers by 2^b . Now we use the algorithm \mathcal{A} to obtain a solution and use the verifiers to test for optimality. In case of FAILURE we increment b by one and iterate until the verifiers conclude optimality of the computed solution.

To analyze the running time of \mathcal{A} we need to estimate W , the largest absolute value of any integer in the stochastic expressions. It is the product of two factors. The first factor, $W_1 = 2^b$, is due to the scaling and depends on the number of revealed bits after the binary point of each coefficient. The second factor, W_2 , corresponds to the integer part of the largest absolute value of any coefficient. This way, $W = W_1W_2$. We have to show that there exists a polynomial $P(N, \phi, \frac{1}{p})$ such that for every $N \in \mathbb{N}, \phi > 0, p \in [0, 1] : \Pr \left[T > P(N, \phi, \frac{1}{p}) \right] \leq p$. The running time of \mathcal{A} is polynomial in W, n, k , and N . As n, k , and N do not depend on the random choice for the coefficients, it suffices to show such a polynomial bound for W . Let us first prove an upper bound on W_2 . Let $r_{j,i}$ be the random variable that is added to the i th coefficient in the j th stochastic expression when perturbing the instance. Recall that the mean of the absolute values of the probability distribution that defines the perturbation model is a constant, denoted by E . It then holds that $W_2 \leq \max_{j \in [k], i \in [n]} |r_{j,i}| + 1$. For every $\alpha \geq 1$,

$$\Pr \left[\max_{j,i} |r_{j,i}| > \alpha E \right] \leq \sum_{j,i} \Pr [|r_{j,i}| > \alpha E] \leq \frac{nk}{\alpha},$$

where the last inequality uses the Markov inequality. Thus, $\Pr [W_2 > \alpha E + 1] \leq nk/\alpha$. Setting $\alpha = 2nk/p$, it holds that $W_2 \leq \frac{2nkE}{p} + 1$ with a probability of at least $1 - p/2$, for every $0 < p < 1$. Next consider the term $W_1 = 2^b$. There are two reasons the certifier can declare the computed solutions suboptimal or infeasible. First, the random instance happens to have a small winner gap or a small loser gap such that the winner is not uniquely determined. By Theorems 9 and 10, the probability for this event is at most $k\phi n^3/2^b$. Second, the feasibility checker reports false negatives due to a small feasibility gap. This happens with a probability of at most $k'\phi n^3/2^b$. Allowing a failure probability of $p/4$ for each event, we obtain $W_1 = 2^b = \frac{k\phi n^3 4}{p}$. Therefore, with a probability of at least $1 - p$, none of these bad events occur and it holds that $W = W_1W_2 \leq \left(\frac{2nkE}{p} + 1 \right) \frac{k\phi n^3 4}{p}$. As E is assumed to be constant and $kn \leq N$, there exists a polynomial $P(N, \phi, \frac{1}{p})$ such that for all $N \in \mathbb{N}, \phi > 0, p \in [0, 1] : \Pr \left[W > P(N, \phi, \frac{1}{p}) \right] \leq p$.

From polynomial smoothed complexity to pseudo-polynomial running time. Finally, we need to show that polynomial smoothed complexity of a binary

optimization problem Π implies the existence of a randomized pseudo-polynomial algorithm for Π . Since we are aiming for a pseudo-polynomial time algorithm, we can assume that all numbers in the stochastic expressions are integers. Let M denote the largest absolute value of these numbers. The idea is to perturb all numbers only slightly such that the perturbation changes the value of each expression by at most $\frac{1}{2}$. To ensure that the set of feasible solutions is not changed by the perturbation, we relax all constraints by $\frac{1}{2}$, i.e., we replace $w^T x \leq t$ by $w^T x \leq t + \frac{1}{2}$ for all stochastic constraints. We then use an algorithm with polynomial smoothed complexity to compute an optimal solution x^* for the perturbed problem. By bounding the error due to the random perturbation, x^* can be shown to be optimal for the original problem as well.

Let us describe the proof in more detail. Our smoothed analysis framework assumes that all numbers in the stochastic expressions fall into the interval $[-1, 1]$ (or $[0, 1]$) before they are perturbed. To adapt our problem to this framework, we first scale all input numbers in the stochastic expressions by M^{-1} and adapt the thresholds accordingly, i.e., $w_1 x_1 + w_2 x_2 + \dots + w_n x_n \leq t$ is replaced by $(w_1/M)x_1 + (w_2/M)x_2 + \dots + (w_n/M)x_n \leq t/M$. Consequently, we have to ensure that the perturbation changes the value of an expression by at most $1/(2M)$. In particular, we will allow only perturbations that change each individual number by at most $1/(2Mn)$. We call such a perturbation *proper*. For the uniform distribution, we could simply set $\phi = 2Mn$. However, we have to deal with arbitrary families of distributions, as defined in our smoothed analysis framework, and they do not necessarily have a finite domain. The idea is to choose ϕ large enough so that a random perturbation is proper with a probability of at least $1/2$. Recall that the perturbation model is described by the density function f with density parameter $\phi = 1$. For other values of ϕ , we scale f appropriately. By our assumptions on f , it holds that $\int |t| f_\phi(t) dt = E/\phi$ for some fixed $E \in \mathbb{R}$. Let r be a random variable following f_ϕ . Setting $\phi = 4n^2 k E M$ and applying the Markov inequality yields $\Pr [|t| > \frac{1}{2nM}] = \Pr [|t| > \frac{2nkE}{\phi}] \leq \frac{1}{2nk}$. Our perturbation draws kn of these random variables. The probability that the perturbation is proper, i.e., the probability that their absolute values are at most $\frac{1}{2nM}$, is $1/2$.

Consider any binary optimization problem Π with polynomial smoothed complexity. Polynomial smoothed complexity implies that the problem admits an algorithm \mathcal{A} whose running time can be bounded polynomially in n and ϕ , with arbitrary large constant probability strictly less than 1. In particular, there exists a polynomial $P(n, \phi)$ such that the probability that the running time exceeds $P(n, \phi)$ is at most $\frac{1}{4}$. We use \mathcal{A} as a subroutine in order to obtain a pseudo-polynomial algorithm. This algorithm works as follows. At first, it generates a perturbation and checks whether it is *proper*. If it is proper, then it runs \mathcal{A} for at most $P(n, \phi)$ time steps. If \mathcal{A} has not finished within this time bound, the algorithm returns FAILURE. Let Q be the event that the perturbation is proper. Observe that for every two events A and B it holds that $\Pr [A \wedge B] \geq \Pr [A] + \Pr [B] - 1$. Therefore, the success probability of our algorithm is

$$\Pr [Q \wedge (T \leq P(n, \phi))] \geq \Pr [Q] - \Pr [T > P(n, \phi)] \geq \frac{1}{4}.$$

The running time of this algorithm is pseudo-polynomial because $\phi = O(Mn^2k)$. Hence, $\Pi_u \in \text{ZPP}$. This completes the proof of Theorem 11. \square

4. Algorithmic applications. Let us illustrate the strength of Theorem 11 by giving algorithmic applications to some well-known optimization problems and comparing these results with previous work on the probabilistic analysis of optimization problems. There has been substantial effort to analyze random instances of the knapsack problem; see, e.g., [5, 6, 13, 17, 18]. The knapsack problem can be seen as the simplest nontrivial binary optimization problem as its feasible region is described by only a single linear constraint. The problem belongs to the class of packing problems, that is, the constraint is of the form $w^T x \leq t$ and the coefficients are assumed to be nonnegative. To our knowledge, the knapsack problem is the only NP-hard optimization problem that was previously known to have polynomial smoothed complexity [5]. The multidimensional knapsack problem is a natural generalization in which there are multiple packing constraints instead of only one. Dyer and Frieze [10] proved that, with constant probability, this problem can be solved in polynomial time if the number of constraints is constant and the coefficients in the constraints as well as in the objective function are chosen uniformly at random from $[0, 1]$. Their result, however, does not yield polynomial average-case complexity as the dependence of the running time on the failure probability is not bounded by a polynomial. The multiple knapsack problem with a constant number of constraints admits a pseudo-polynomial algorithm [15]. Hence, Theorem 11 implies polynomial smoothed and, therefore, also polynomial average-case complexity for this problem. Moreover, the pseudo-polynomial algorithm also works for general 0/1 integer programming with any fixed number of constraints, i.e., when extending the domain of the coefficients to negative numbers. Therefore, this class of problems has polynomial smoothed complexity as well. This holds even if one constraint or the objective function is assumed to be adversarial. Furthermore, it follows from Theorem 11 that, unless $ZPP = NP$, general 0/1 integer programming as well as the multiple knapsack problem with an unbounded number of constraints have no polynomial smoothed complexity as they are strongly NP-hard.

It is important to interpret this result in the right way. It does not prove that all problems that can be formulated as a 0/1-integer program (with a constant number of constraints) have polynomial smoothed complexity. When expressing a problem as a 0/1 integer program, some numbers of the input usually appear in more than one constraint. Furthermore, the constraint matrix is usually sparse, containing many zero entries. Perturbing the constraints independently, as assumed by our smoothed analysis framework, destroys the structure of the problem. So instead of averaging over “similar” instances of the given problem, we would average over instances of some more general problem that might be easier to solve on average. We will see in section 6 how to strengthen our analysis by allowing zero entries in the constraints which are explicitly not perturbed. This way we are able to extend our analysis to problems with sparse constraint vectors.

4.1. Scheduling problems. The problem of scheduling to minimize the weighted number of tardy jobs is defined by n jobs each of which has a processing time p_i , a due date d_i , and a penalty c_i , which has to be paid if job i has not been finished at due date d_i . The jobs shall be scheduled on a single machine such that the sum of the penalties is minimized. In terms of n binary variables x_1, \dots, x_n , the objective is to minimize $c^T x$, where $x_i = 1$ if job i is not finished in time. Observe that the problem is essentially solved once these binary variables are determined as we can assume without loss of generality that an optimal schedule executes the selected jobs in the order of nondecreasing due dates. Notice that the feasible region is completely

determined by the processing times and the due dates. The objective, however, is a linear function $c^T x$. Using dynamic programming, the problem can be solved in time $O(n^2C)$, where C denotes the largest penalty for any job [24]. Hence, the problem has polynomial smoothed complexity for stochastic penalties.

4.2. Multicriteria optimization problems. If several criteria shall be optimized simultaneously, then usually one of them is declared to be the objective function, and the others are formulated in the form of a constraint with a given threshold. Often when a single-criteria optimization problem is polynomial, the problem becomes NP-hard when adding another criteria in the form of a linear constraint; examples for such problems are shortest path, spanning tree, or matching [21, 12, 23]. Theorem 11 enables us to prove polynomial smoothed complexity for such multicriteria problems as follows. The problems listed above have algorithms with pseudo-polynomial running time [23, 3, 22] solving the “exact version” of these problems, i.e., given an integer k and an instance of these problems, one can compute a solution with objective value exactly k in pseudo-polynomial time. Using standard coding techniques (see, e.g., [28]) a pseudo-polynomial algorithm for the exact single-criteria decision problem implies a pseudo-polynomial algorithm for its multicriteria optimization variant. Combining this observation with Theorem 11 yields the following result.

COROLLARY 12. *Let Π be a binary optimization problem with a single linear objective function. Suppose the exact version of Π admits an algorithm with pseudo-polynomial running time. Assume we deal with multicriteria variants of Π by choosing one criterion as the objective function and bounding the remaining criteria by appropriate thresholds. Then any multicriteria variant of Π with a constant number of criteria has polynomial smoothed complexity, provided that all criteria are stochastic.*

A similar approach was used in [23] to derive approximation schemes for multiobjective optimization problems. The corollary implies polynomial smoothed complexity for the multicriteria variants of shortest path, spanning tree, and matching. One does not always need to assume that all criteria are of stochastic nature. For example, the bicriteria variant of the shortest path problem, i.e., the constrained shortest path problem, can be solved in pseudo-polynomial time with respect to the objective function or with respect to the additional constraint. By Theorem 11, the constrained shortest path problem has polynomial smoothed complexity even if only the objective function or only the additional constraint is stochastic.

5. Expected polynomial running time. The main advantage of our definition of polynomial smoothed complexity is its robustness under different machine models. Besides, it allows a nice characterization of binary optimization problems under random inputs in terms of the problems’ worst-case complexity. However, the guarantee on the running time provided by polynomial smoothed/average-case complexity is weaker than the guarantee that the expected running time is polynomial. Making additional assumptions, we can use our analysis to conclude expected polynomial running time for certain binary optimization problems.

We use again our adaptive rounding scheme, which increases the precision of the stochastic input numbers until the computed solution is certified to be optimal. We assume that the running time of the last iteration of this meta-algorithm dominates the cumulative running time of all previous iterations. In fact, it suffices if all previous iterations have cumulative running time of at most a factor n^l larger than the running time of the last iteration, for some constant $l \in \mathbb{R}$. In order to obtain expected polynomial running time under random perturbations, one needs an algorithm whose

running time is “pseudo-linear” instead of only pseudo-polynomial with respect to the stochastic coefficients, i.e., the running time must be bounded linearly in the size of the unary encoding length of the integer coefficients. The only parameter in our running time bound that depends on the random perturbations is W , the largest absolute value of any (rounded and scaled) coefficient appearing in a stochastic expression.

We use the notation from the proof of Theorem 11. We divide W into two parts $W = W_1W_2$, where W_1 is due to the scaling by factor $2^b = W_1$ and depends on b , the number of revealed bits after the binary point of each coefficient. The second factor W_2 corresponds to the integer part of the largest absolute value of any coefficient. In the proof of Theorem 11 we have shown the following bounds on the random variables W_1 and W_2 :

$$\begin{aligned} \Pr [W_1 > 2^b] &\leq 2k\phi n^3/2^b, & \text{for any } b \in \mathbb{N}, \text{ and} \\ \Pr [W_2 > \alpha E + 1] &\leq nk/\alpha, & \text{for any } \alpha \in \mathbb{R}_{>0}. \end{aligned}$$

Since 2^b as well as α can grow linearly with the reciprocal of the failure probability, $W \approx 2^b\alpha E$ can grow quadratically. Hence, these bounds do not allow one to conclude a polynomial expected running time. Therefore, we will restrict the choice for the perturbation model by allowing only probability distributions whose tail function exhibits an exponential decay. More precisely, we assume that if X is drawn according to perturbation model f , then there exists some $E \in \mathbb{R}_{\geq 0}$, such that $\Pr [|X| > \alpha E] \leq 2^{-\alpha}$, for every $\alpha \geq 2$. For example, the Gaussian and exponential distribution have this property as well as all distributions with finite domain.

In the following analysis we exploit the fact that it is very unlikely that any of the coefficients in the stochastic expressions is much larger than E . Define event \mathcal{E} by $(W_2 \leq n \log(nk)E + 1) \wedge (W_1 \leq 2^n)$. As $\Pr [W_1 > 2^n] \leq 2k\phi n^3/2^n$ and $\Pr [W_2 > n \log(nk)E + 1] \leq kn \Pr [\max_{j,i} |r_{i,j}| > n \log(nk)E] \leq 2^{-n}$, it follows that $\Pr [-\mathcal{E}] \leq O(k\phi n^3/2^n)$. Assume that there exists an algorithm with pseudo-linear running time bound of $T \leq N^l W$, for some constant $l \in \mathbb{R}$. In case of $-\mathcal{E}$, we use a brute force enumeration of all 2^n possible 0/1-vectors which takes time $2^n P(N)$ for some polynomial $P(N)$. Additionally, we contribute the time for the unsuccessful adaptive rounding scheme, which is at most $N^l 2^n (n \log(nk)E + 1)$, to the enumeration process. Hence, the total running time is $O(2^n P(N))$ for some other polynomial $P(N)$. Next define random variable

$$W'_1 = \begin{cases} W_1 & \text{in case of } \mathcal{E}, \\ 0 & \text{otherwise.} \end{cases}$$

The expectation of W'_1 is

$$\mathbf{E} [W'_1] = \sum_{i=0}^n \Pr [W'_1 = 2^i] 2^i \leq \sum_{i=0}^n \Pr [W_1 \geq 2^i] 2^i \leq \sum_{i=0}^n \frac{4\phi kn^3}{2^i} 2^i = O(\phi kn^4).$$

Let T denote the running time of our algorithm. As $\mathbf{E} [W'_1] = \Pr [\mathcal{E}] \mathbf{E} [W'_1 | \mathcal{E}]$,

$$\begin{aligned} \mathbf{E} [T] &\leq \Pr [\mathcal{E}] \cdot \mathbf{E} [N^l W_1 W_2 | \mathcal{E}] + \Pr [-\mathcal{E}] \cdot 2^n P(N) \\ &\leq N^l (n \log(nk)E + 1) \cdot \Pr [\mathcal{E}] \mathbf{E} [W'_1 | \mathcal{E}] + \Pr [-\mathcal{E}] \cdot 2^n P(N) \\ &\leq N^l (n \log(nk)E + 1) \mathbf{E} [W'_1] + O(k\phi n^3/2^n) \cdot 2^n P(N) \\ &= O(N^l (n \log(nk)E + 1) \phi kn^4 + k\phi n^3 P(N)). \end{aligned}$$

COROLLARY 13. *Assume the perturbation model f satisfies $\Pr[|X| > \alpha E] \leq 2^{-\alpha}$, for some $E \in \mathbb{R}_{\geq 0}$ and all $\alpha \geq 2$. If a binary optimization problem has pseudo-polynomial running time $O(\text{poly}(N)W)$, then this problem allows an algorithm with expected polynomial running time.*

Such pseudo-linear algorithms exist on a uniform RAM, e.g., for the knapsack problem [8], the problem of scheduling to minimize weighted tardiness [24] or the constrained shortest path problem [16]. Hence, assuming the uniform RAM model, all these problems admit algorithms with expected polynomial running time under random perturbations. Observe, however, that the requirement of a pseudolinear running time is a strong restriction for binary optimization problems with $k > 1$ stochastic constraints, as standard approaches usually exhibit a running time of $\text{poly}(n)W^k$. In case of dynamic programming, for example, every additional constraint will add another dimension to the dynamic programming table.

6. Zero-preserving perturbations. One criticism of the smoothed analysis of the simplex algorithm is that the additive perturbations destroy the zero structure of an optimization problem, as it replaces zeros with small values; see also the discussion in [29]. The same criticism applies to the zero structure in binary programs. It turns out, however, that our probabilistic analysis in section 2 is robust enough to allow the preservation of zero entries. In particular, we can extend our semirandom input model introduced in section 1.1 by allowing the coefficients in the stochastic expressions to be fixed to zero instead of being a random variable. In the model of smoothed analysis, this corresponds to strengthening the adversary by avoiding the perturbation of these zero coefficients.

Consider the expression $w^T x$ and let Z be the set of indices i with w_i fixed to zero. We call two solutions $x, x' \in \mathcal{S} \subseteq \{0, 1\}^n$ equivalent if they differ only in positions contained in Z , e.g., if $x_i \neq x'_i \Rightarrow i \in Z$ holds. This way, Z defines equivalence classes on \mathcal{S} with respect to the expression $w^T x$. Clearly, $w^T x$ evaluates to the same value for all solutions within the same equivalence class.

First we consider the separation lemma. Observe that only the highest ranked solution in each equivalence class is relevant for the loser and feasibility gap. This is because the winner and the minimal loser are Pareto optimal solutions. As all solutions within an equivalence class have the same weight, only the highest ranked solution of this class can become Pareto optimal. For the purpose of analysis, we can remove virtually all solutions from \mathcal{S} that are not ranked highest within its equivalence class. This way, we can ignore variables x_i with $i \in Z$ and apply the separation lemma as before.

A similar argument can be used to show that the generalized isolating lemma stays valid with respect to equivalence classes, that is, the winner gap is defined to be the difference in objective value between the best and the second best equivalence class. However, it might be very likely that there are many optimal solutions, as the winning equivalence class might have many solutions. Notice that this affects the procedure that certifies optimality for stochastic objective functions which is described in the proof of Theorem 11. In particular, the two solutions x' and x'' , which are optimal with respect to the rounded cost vectors $\lfloor c \rfloor$ and \bar{c} , only have to be in the same equivalence class to certify optimality, which can be checked easily.

6.1. Algorithmic application. Using zero-preserving perturbations, we can apply our analysis to the general assignment problem (GAP), which is defined as follows:

$$\begin{aligned} \max \quad & \left(\sum_{i=1}^k \sum_{j=1}^n p_{ij} x_{ij} \right) \quad \text{subject to} \quad \sum_{j=1}^n w_{ij} x_{ij} \leq c_i, \quad i = 1, \dots, k, \\ & \sum_{i=1}^k x_{ij} = 1, \quad j = 1, \dots, n, \\ & x_{ij} \in \{0, 1\} \quad \text{for all } i \in [k], j \in [n]. \end{aligned}$$

Intuitively, the goal is to pack n items into k bins with capacities c_1, \dots, c_k . Packing item $j \in [n]$ into bin $i \in [k]$ occupies w_{ij} units of the capacity of bin i and yields a profit of p_{ij} . Even for two bins, the GAP problem does not allow a fully polynomial time approximation scheme (FPTAS), unless $P = NP$ [20]. In fact, the problem is strongly NP-hard for an unbounded number of bins. If, however, the number of bins is a constant, then the GAP can be solved in pseudo-polynomial time using standard dynamic programming as follows. Let $T(j, W_1, \dots, W_k)$ denote the maximum profit over solutions that use only items $\{1, \dots, j\}$ such that the cumulative weight of all items in bin i is exactly W_i , for all $i \in [k]$. All nontrivial entries can be computed by the following recursion:

$$T(j, W_1, \dots, W_k) = \max_{i \in [k]} \{T(j-1, V_1, \dots, V_k) + p_{ij} \mid V_i = W_i - w_{ij}, V_l = W_l \forall l \neq i\}.$$

Let W denote the largest weight of the instance, i.e., $W = \max_{j \in [n]} \max_{i \in [k]} w_{ij}$. Then the size of the dynamic programming table is at most $n(nW)^k$. Hence, the running time is at most $kn(nW)^k$.

How does the GAP problem fit into our framework? We have a vector of kn binary variables $(x_{11}, x_{12}, \dots, x_{1n}, x_{21}, x_{22}, \dots, x_{2n}, \dots, x_{k1}, x_{k2}, \dots, x_{kn})$ and the corresponding vector of profits. The weight constraint for bin $l \in [k]$ uses the weight vector with all entries w_{ij} , $i \neq l$, set to zero. These entries are declared to be fixed to zero, so they are not touched by the perturbation. This way, each w_{ij} appears in just one constraint and a perturbation of the profit and the constraint vectors has the same effect as perturbing each p_{ij} and w_{ij} individually. This yields the following corollary.

COROLLARY 14. *The general assignment problem with a constant number of constraints/bins has polynomial smoothed complexity if weights and profits are randomly perturbed.*

The discussion above reveals a limitation of our analysis. For this purpose, consider the multiple knapsack problem, a special case of the GAP problem, where the weight of an item is the same for all bins, i.e., $w_{ij} = w_j$ for all $i \in [k]$. Using the program formulation of the GAP problem, the weights w_j reoccur in the different constraints for the different knapsacks. The perturbation of the constraints, however, have to be independent such that the perturbed instance is, in general, not a multiple knapsack instance any more. We observe that for a sensible application of our analysis, each variable that describes the problem instance must appear at most once in the stochastic constraints of the linear program formulation. Or, seeing it another way, the coefficients in the stochastic constraints must allow independent perturbations without destroying the structure of the problem.

7. Other aspects.

7.1. Smoothed complexity and approximation schemes. If a binary optimization problem Π has polynomial smoothed complexity when perturbing only the coefficients in the objective function, then Π also admits an absolute fully polynomial

randomized approximation scheme. The idea is to perturb the instance only slightly, resulting in a very similar optimal objective function value. The set of feasible solutions is not affected by the perturbation. In order to bound the running time, we can exploit the positive influence of the added randomness. Let us give some more details.

Given a (worst-case) instance of Π , the coefficients in the objective function are normalized such that the largest absolute value is 1. Subsequently, the normalized coefficients are perturbed using the uniform perturbation model with parameter ϕ . As each coefficient changes by at most $1/\phi$, the difference in the objective value of any two solutions can change by at most n/ϕ due to this perturbation. Hence, the optimal solution of the perturbed instance is, with respect to the unperturbed objective function, at most n/ϕ away from the optimum. According to the definition of polynomial smoothed complexity, we can fix some polynomial bound B such that the running time of the algorithm is smaller than B with a probability of at least $1/2$. In order to obtain polynomial expected running time we generate perturbations and run the algorithm for at most B time steps. If the algorithm has not finished, we generate a new perturbation and try again. As the perturbations are independent, one needs 2 iterations on average until a solution is found. Hence, the expected running time is $2B$.

The approximation scheme allows only one to bound the absolute error. If, however, the optimum of the normalized instance can be lower bounded by some polynomial in $1/n$, then we obtain a randomized FPTAS.

The opposite direction works as well. Suppose Π is a binary optimization problem admitting a fully polynomial approximation scheme. It is well known that such an approximation scheme can be transformed into a pseudo-polynomial algorithm [11]. The running time of this algorithm is pseudo-polynomial only with respect to the coefficients in the objective function. Thus, Theorem 11 shows that Π has polynomial smoothed complexity when perturbing only the objective function.

This shows that for the class of problems considered, there exists a correlation between problems that allow an FPTAS and problems with polynomial smoothed complexity for stochastic linear objective functions.

Observe that these results rely solely on the analysis of the winner gap. The results based on the loser and feasibility gaps yield polynomial smoothed complexity for problems for which no fully polynomial approximation scheme is known, such as the constrained spanning tree problem, or even for problems that cannot be approximated within any polynomial-time computable factor, such as 0/1 programming with a constant number of constraints. In fact, the problems to which these gaps apply are exactly those problems admitting a bicriteria approximation scheme. In other words, our results show that the impact of randomly perturbing the objective function and/or the constraints on the computational complexity of a problem is comparable with the impact of relaxing the same set of expressions. The advantage of smoothed analysis over (bicriteria) approximation schemes, however, is that it yields optimal and feasible instead of almost optimal and almost feasible solutions.

7.2. Euclidean optimization problems. Upon first view, the Euclidean variants of TSP and Steiner tree might look like interesting candidates for problems with polynomial smoothed complexity. Karp [14], in a seminal work on the probabilistic analysis of algorithms, studied the TSP problem in a model in which n points are drawn uniformly and independently from the unit square. A natural extension of this model would be to assume that first an adversary chooses points in the unit cube

or ball, and then one applies a multidimensional Gaussian perturbation. We claim, however, that neither Euclidean TSP nor Steiner tree do have polynomial smoothed complexity, since the perturbation does not change the set of feasible solutions. The change in the objective function due to the perturbation depends at most linearly on the magnitude of the perturbation. Since the optimal objective value is $\Omega(1)$, we could obtain a fully polynomial randomized approximation scheme for worst-case instances, which is widely believed not to exist. Thus, one needs to give up either the requirement that the running time is polynomial in $1/\epsilon$ or the requirement that the running time is polynomial in ϕ . Under the first relaxation the problem has been solved by Arora [1]. The question whether there exist polynomial time algorithms for Euclidean TSP or the Steiner tree under perturbations with a fixed density parameter ϕ or in the uniform input model of Karp [14] is open.

7.3. Relationship to condition numbers. In order to obtain a finer analysis of algorithms than that provided by worst-case complexity, one might seek to find a way for distinguishing hard problem instances from easy ones. A natural approach is to find a quantity indicating the difficulty of solving a problem instance. In numerical analysis and operations research it is common to bound the running time of an algorithm in terms of a *condition number* of its input. The condition number is typically defined to be the sensitivity of the solution for a problem instance to slight perturbations of the input. For example, Renegar [25, 26, 27] presents a variant of the primal interior point method and describes its running time as a function of the condition number. Remarkably, his running time bound depends only logarithmically on the condition number. Dunagan, Spielman, and Teng [9] study this condition number in the smoothed analysis framework. Assuming Gaussian ϕ -perturbations, Renegar's condition number can be bounded by a function that is polynomial in ϕ . Thus, the running time of Renegar's interior point method depends only logarithmically on the density parameter ϕ . In contrast, the running time bound of the simplex algorithm presented by Spielman and Teng in [29] is polynomial in ϕ .

In [30], Spielman and Teng propose to extend the condition number concept toward discrete optimization problems in order to assist the smoothed analysis of such problems. As a natural definition for the condition number of a discrete function, they suggest *the reciprocal of the minimum distance of an input to one on which the function has a different value*. In fact, the minimum of winner, loser, and feasibility gap is a lower bound on the amount by which the coefficients of a binary optimization problem need to be altered so that the winner, i.e., the solution taking the optimal value, changes. Let us define the reciprocal of this minimum to be the *condition number for binary optimization problems*. This allows us to summarize our analysis in an alternative way. Our probabilistic analysis in section 2 shows that the condition number is bounded polynomially in the density parameter ϕ . Furthermore, in section 3, we proved that a problem with pseudo-polynomial worst-case complexity admits an algorithm whose running time is bounded polynomially in the condition number. Combining these results, we obtained algorithms whose smoothed complexity depends upon a polynomial fashioned on the density parameter ϕ . Let us remark that this kind of dependence on ϕ is best possible for NP-hard optimization problems, unless there is a subexponential time algorithm for NP-complete problems. In particular, a running time bound logarithmic in ϕ for an NP-hard optimization problem such as the knapsack problem would imply a randomized algorithm with polynomial worst-case complexity: Perturb all input numbers using a suitable distribution with sufficiently small density such that the identity of the optimal solution is not affected and com-

pute the optimal solution for the perturbed instance. For example, perturbing the profit values of the knapsack problem with a uniform distribution with exponentially small density parameter ϕ , the optimal solution of the perturbed instance is optimal for the original instance as well. Obviously, a smoothed analysis with running time logarithmic in ϕ would thus imply a polynomial running time for the NP-hard knapsack problem.

Acknowledgments. We thank Uriel Feige for helpful comments on an early version of this manuscript and an anonymous reviewer for several suggestions that improved the presentation of our analysis.

REFERENCES

- [1] S. ARORA, *Polynomial time approximation schemes for Euclidean TSP and other geometric problems*, in Proceedings of the 37th Annual Symposium on Foundations of Computer Science (FOCS), Burlington, VT, IEEE Press, Los Alamitos, CA, 1996, pp. 2–11.
- [2] C. BANDERIER, R. BEIER, AND K. MEHLHORN, *Smoothed analysis of three combinatorial problems*, in Proceedings of the 28th International Symposium on Mathematical Foundations of Computer Science (MFCS-2003), vol. 2747, Bratislava, Slovak Republic, Springer, LNCS Series, 2003, pp. 198–207.
- [3] F. BARAHONA AND W. R. PULLEYBLANK, *Exact arborescences, matchings, and cycles*, Discrete Appl. Math., 16 (1987), pp. 91–99.
- [4] L. BECCHETTI, S. LEONARDI, A. MARCHETTI-SPACCAMELA, G. SCHÄFER, AND T. VREDEVELD, *Average case and smoothed competitive analysis of the multi-level feedback algorithm*, in Proceedings of the Forty-Fourth Annual IEEE Symposium on Foundations of Computer Science, IEEE Computer Society, Cambridge, MA, 2003, pp. 462–471.
- [5] R. BEIER AND B. VÖCKING, *Random knapsack in expected polynomial time*, in Proceedings of the 35th Annual ACM Symposium on Theory of Computing (STOC-2003), San Diego, CA, ACM Press, New York, 2003, pp. 232–241.
- [6] R. BEIER AND B. VÖCKING, *Probabilistic analysis of knapsack core algorithms*, in Proceedings of the 15th Annual Symposium on Discrete Algorithms (SODA-2004), New Orleans, LA, SIAM, Philadelphia, 2004, pp. 461–470.
- [7] V. DAMEROW, F. M. AUF DER HEIDE, H. RÄCKE, C. SCHEIDELER, AND C. SOHLER, *Smoothed motion complexity*, in Proceedings of the 11th European Symposium on Algorithms, Budapest, Hungary, Springer, Berlin, 2003, pp. 161–171.
- [8] G. B. DANTZIG, *Discrete variable extremum problems*, Operations Research, 5 (1957), pp. 266–277.
- [9] J. DUNAGAN, D. SPIELMAN, AND S.-H. TENG, *Smoothed analysis of Renegar’s condition number for linear programming*, available at <http://arxiv.org/abs/cs.DS/0302011>. Submitted for publication, 2003.
- [10] M. DYER AND A. M. FRIEZE, *Probabilistic analysis of the multidimensional knapsack problem*, Math. Oper. Res., 14 (1989), pp. 162–176.
- [11] M. R. GAREY AND D. S. JOHNSON, *Computers and Intractability*, Freeman and Company, San Francisco, 1979.
- [12] M. X. GOEMANS AND R. RAVI, *The constrained minimum spanning tree problem*, in Proceedings of the Fifth Scandinavian Workshop on Algorithm Theory, Reykjavik, Iceland, Lecture Notes in Comput. Sci. 1097, Springer, 1996, pp. 66–75.
- [13] A. GOLDBERG AND A. MARCHETTI-SPACCAMELA, *On finding the exact solution to a zero-one knapsack problem*, in Proceedings of the 16th ACM Symposium on Theory of Computing (STOC), Washington, D.C., ACM Press, New York, 1984, pp. 359–368.
- [14] R. M. KARP, *Probabilistic analysis of partitioning algorithms for the traveling salesman problem*, Math. Oper. Res., 2 (1977), pp. 209–224.
- [15] H. KELLERER, U. PFERSCHY, AND D. PISINGER, *Knapsack Problems*, Springer, Berlin, 2004.
- [16] E. L. LAWLER, *Combinatorial Optimization: Networks and Matroids*, Holt, Rinehart, and Winston, New York, 1976.
- [17] G. S. LUEKER, *On the average difference between the solutions to linear and integer knapsack problems*, Applied Probability—Computer Science. The Interface, Vol. 1, Progr. Comput. Sci., 2 (1982), pp. 489–504.
- [18] G. S. LUEKER, *Average-case analysis of off-line and on-line knapsack problems*, J. Algorithms, 19 (1998), pp. 277–305.

- [19] G. S. LUEKER, *Exponentially small bounds on the expected optimum of the partition and subset sum problems*, Random Structures Algorithms, 12 (1998), pp. 51–62.
- [20] S. MARTELLO AND P. TOTH, *Knapsack Problems. Algorithms and Computer Implementations*, Wiley, Chichester, 1990.
- [21] K. MEHLHORN AND M. ZIEGELMANN, *Resource constrained shortest paths*, in Proceedings of the 8th Annual European Symposium (ESA-00), Saarbrücken, Germany, Lecture Notes in Comput. Sci., 1879, Springer, Berlin, 2000, pp. 326–337.
- [22] K. MULMULEY, U. VAZIRANI, AND V. VAZIRANI, *Matching is as easy as matrix inversion*, Combinatorica, 7 (1987), pp. 105–113.
- [23] C. H. PAPADIMITRIOU AND M. YANNAKAKIS, *The complexity of tradeoffs and optimal access of web sources*, in Proceedings of the IEEE Symposium on Foundations of Computer Science (FOCS), Redondo Beach, CA, IEEE Computer Society, Cambridge, MA, 2000, pp. 86–92.
- [24] R. G. PARKER, *Deterministic Scheduling Theory*, Chapman and Hall, London, 1995.
- [25] J. RENEGAR, *Some perturbation theory for linear programming*, Math. Programming, 65 (1994), pp. 73–91.
- [26] J. RENEGAR, *Incorporating condition measures into the complexity theory of linear programming*, SIAM J. Optim., 5 (1995), pp. 506–524.
- [27] J. RENEGAR, *Linear programming, complexity theory, and elementary functional analysis*, Math. Programming, 70 (1995), pp. 279–351.
- [28] H. M. SALKIN, *Integer Programming*, Addison-Wesley, Reading, MA, 1975.
- [29] D. A. SPIELMAN AND S.-H. TENG, *Smoothed analysis of algorithms: Why the simplex algorithm usually takes polynomial time*, in Proceedings of the 33rd ACM Symposium on Theory of Computing (STOC), Heraklion, Crete, Greece, ACM, New York, 2001, pp. 296–305.
- [30] D. A. SPIELMAN AND S.-H. TENG, *Smoothed analysis: Motivation and discrete models*, in Proceedings of WADS, Ottawa, Ontario, Canada, Lecture Notes in Comput. Sci., 2748, Springer, Berlin, 2003, pp. 256–270.

SPECTRAL PARTITIONING, EIGENVALUE BOUNDS, AND CIRCLE PACKINGS FOR GRAPHS OF BOUNDED GENUS*

JONATHAN A. KELNER[†]

Abstract. In this paper, we address two long-standing questions about finding good separators in graphs of bounded genus and degree:

1. It is a classical result of Gilbert, Hutchinson, and Tarjan [*J. Algorithms*, 5 (1984), pp. 391–407] that one can find asymptotically optimal separators on these graphs if given both the graph *and* an embedding of it onto a low genus surface. Does there exist a simple, efficient algorithm to find these separators, given only the graph and not the embedding?
2. In practice, spectral partitioning heuristics work extremely well on these graphs. Is there a theoretical reason why this should be the case?

We resolve these two questions by showing that a simple spectral algorithm finds separators of cut ratio $O(\sqrt{g/n})$ and vertex bisectors of size $O(\sqrt{gn})$ in these graphs, both of which are optimal. As our main technical lemma, we prove an $O(g/n)$ bound on the second smallest eigenvalue of the Laplacian of such graphs and show that this is tight, thereby resolving a conjecture of Spielman and Teng. While this lemma is essentially combinatorial in nature, its proof comes from continuous mathematics, drawing on the theory of circle packings and the geometry of compact Riemann surfaces.

Key words. bounded genus, circle packing, graph separators, Laplacian, partitioning, spectral partitioning

AMS subject classifications. 05C85, 68R10, 68W40, 68Q25, 68W05

DOI. 10.1137/S0097539705447244

1. Introduction. Spectral methods have long been used as a heuristic in graph partitioning. They have had tremendous experimental and practical success in a wide variety of scientific and numerical applications, including mapping finite element calculations on parallel machines [24, 29], solving sparse linear systems [7, 8], partitioning for domain decomposition [8], and VLSI circuit design and simulation [6, 16, 3]. However, it is only recently that people have begun to supply formal justification for the efficacy of these methods [15, 25]. In [25], Spielman and Teng used the results of Mihail [22] to show that the quality of the partition produced by the application of a certain spectral algorithm to a graph can be established by proving an upper bound on the Fiedler value of the graph (i.e., the second smallest eigenvalue of its Laplacian). They then provided an $O(1/n)$ bound on the Fiedler value of a *planar* graph with n vertices and bounded maximum degree. This showed that spectral methods can produce a cut of ratio $O(\sqrt{1/n})$ and a vertex bisector of size $O(\sqrt{n})$ in a bounded degree planar graph.

In this paper, we use the theory of circle packings and conformal mappings of compact Riemann surfaces to generalize these results to graphs of positive genus. We prove that the Fiedler value of a genus g graph of bounded degree is $O(g/n)$ and demonstrate that this is asymptotically tight, thereby resolving a conjecture of Spielman and Teng. We then apply this result to obtain a spectral partitioning algorithm that finds separators whose cut ratios are $O(\sqrt{g/n})$ and vertex bisectors of size $O(\sqrt{gn})$, both of which are optimal. To our knowledge, this provides the only

*Received by the editors September 30, 2004; accepted for publication (in revised form) April 18, 2005; published electronically February 21, 2006.

<http://www.siam.org/journals/sicomp/35-4/44724.html>

[†]Computer Science and Artificial Intelligence Laboratory, Massachusetts Institute of Technology, Cambridge, MA 02139 (kelner@mit.edu).

truly practical algorithm for finding such separators and vertex bisectors for graphs of bounded genus and degree. While there exist other asymptotically fast algorithms for this, they all rely on being given an embedding of the graph in a genus g surface (e.g., [14]). It is not always the case that we are given such an embedding, and computing one is quite difficult. (In particular, computing the genus of a graph is NP-hard [27], and the best known algorithms for constructing such an embedding are either $n^{O(g)}$ [12] or polynomial in n but doubly exponential in g [11]. Mohar has found an algorithm that depends only linearly on n [21], but it has an uncalculated and very large dependence on g .) The excluded minor algorithm of Alon, Seymour, and Thomas [2] does not require an embedding of the graph, but the separators that it produces are not asymptotically optimal.

The question of whether there exists an efficient algorithm for providing asymptotically optimal cuts without such an embedding was first posed twenty years ago by Gilbert, Hutchinson, and Tarjan [14].¹ We resolve this question here, as our algorithm proceeds without any knowledge of an embedding of the graph, instead relying only on simple matrix manipulations of the adjacency matrix of the graph. While the analysis of the algorithm requires some somewhat involved mathematics, the algorithm itself is quite simple, and it can be implemented in just a few lines of Matlab code. In fact, the algorithm is only a slight modification of the spectral heuristics for graph partitioning that are widely deployed in practice without any theoretical guarantees.

We believe that the techniques that we employ to obtain our eigenvalue bounds are of independent interest. To prove these bounds, we make what is perhaps the first real use of the theory of circle packings and conformal mappings of positive genus Riemann surfaces in the computer science literature. This is a powerful theory, and we believe that it will be useful for addressing other questions in spectral and topological graph theory.

The structure of the paper is as follows. In section 2, we provide the necessary background in graph theory and spectral partitioning, and we state our main results. In section 3, we provide a brief outline of our proof techniques. In section 4, we review the basic theory of circle packings on compact Riemann surfaces. We then use this theory in sections 5 and 6 to prove our main results.

2. Background in graph theory and spectral partitioning. In this section we provide the basic definitions and results from graph theory and spectral partitioning that we shall require in what follows.

2.1. Graph theory definitions. Throughout the remainder of this paper, let $G = (V, E)$ be a finite, connected, undirected graph with n vertices, m edges, and no loops. In this section, we shall define two objects associated with G : its *Laplacian* and its *genus*.

Let the *adjacency matrix* $A(G)$ be the $n \times n$ matrix whose (i, j) th entry equals 1 if $(i, j) \in E$ and equals 0 otherwise. Let $D(G)$ be the $n \times n$ diagonal matrix whose i th diagonal entry equals the degree of the i th vertex of G .

DEFINITION 2.1. *The Laplacian $L(G)$ is the $n \times n$ matrix given by*

$$L(G) = D(G) - A(G).$$

Since $L(G)$ is symmetric, it is guaranteed to have an orthonormal basis of real eigenvectors and exclusively real eigenvalues. Let $\lambda_1 \leq \lambda_2 \leq \dots \leq \lambda_n$ be the eigenval-

¹Djidjev claimed in a brief note to have such an algorithm [10], but it has never appeared in the literature.

ues of $L(G)$, and let v_1, \dots, v_n be a corresponding orthonormal basis of eigenvectors. For any G , the all-ones vector will be an eigenvector of eigenvalue 0. It is not difficult to see that all of the other eigenvalues will always be nonnegative, so that $v_1 = (1, \dots, 1)^T$ and $\lambda_1 = 0$.

There has been a great deal of work relating the eigenvalues of $L(G)$ to the structure of G . In the present paper, we shall concern ourselves exclusively with λ_2 , also known as the *algebraic connectivity* or *Fiedler value* of G . We call the vector v_2 the *Fiedler vector* of G . As we shall see in section 2.2, the Fiedler value of a graph is closely related to how well connected the graph is.

A different measure of the connectivity of a graph is provided by its *genus*, which measures the complexity of the simplest orientable surface on which the graph can be embedded so that none of its edges cross. Standard elementary topology provides a full classification of the orientable surfaces without boundary. Informally, they are all obtained by attaching finitely many “handles” to the sphere, and they are fully topologically classified (i.e., up to homeomorphism) by the number of such handles. This number is called the *genus* of the surface. The genus 0, 1, 2, and 3 surfaces are shown in Figure 2.1.

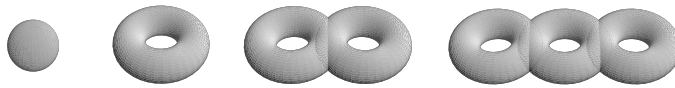


FIG. 2.1. *The surfaces of genus 0, 1, 2, and 3.*

DEFINITION 2.2. *The genus g of a graph G is the smallest integer such that G can be embedded on a surface of genus g without any of its edges crossing one another.*

In particular, a planar graph has genus 0. By making a separate handle for each edge, it is easy to see that $g = O(m)$, where m is the number of edges in G .

Using these definitions, we can now state our main technical result, as follows.

THEOREM 2.3. *Let G be a graph of genus g and bounded degree. Its Fiedler value obeys the inequality*

$$\lambda_2 \leq O\left(\frac{g}{n}\right),$$

and this is asymptotically tight.

The constant in this bound depends on the degree of the graph. The proof that we provide yields a polynomial dependence on the degree, but no effort is made to optimize this polynomial. Finding the optimal such dependence is an interesting open question.

2.2. Spectral partitioning. We recall that a *partition* of a graph G is a decomposition $V = A \cup \bar{A}$ of the vertices of G into two disjoint subsets. For such a partition, we let $\delta(A)$ be the set of edges (i, j) such that $i \in A$ and $j \in \bar{A}$, and we call $|\delta(A)|$ the *cut size* of our partition. The *ratio* of our partition is defined to be

$$\phi(A) = \frac{|\delta(A)|}{\min(|A|, |\bar{A}|)}.$$

If our partition splits the graph into two sets that differ in size by at most one, we call it a *bisection*.

Spectral methods aim to use the Fiedler vector to find a partition of the graph with a good ratio. A theorem that begins to address why these work was proven by Mihail and restated in a more applicable form by Spielman and Teng, as follows.

THEOREM 2.4 (see [22, 25]). *Let G have maximum degree Δ . For any vector x that is orthogonal to the all-ones vector, there is a value s so that the partition of G into $\{i : x_i \leq s\}$ and $\{i : x_i > s\}$ has ratio at most*

$$\sqrt{2\Delta \frac{x^T L(G)x}{x^T x}}.$$

If x is an eigenvector of $L(G)$, the fraction $\frac{x^T L(G)x}{x^T x}$ is equal to its eigenvalue. So, if we find the eigenvector with eigenvalue λ_2 , we will thus quickly be able to find a partition of ratio $\sqrt{2\Delta\lambda_2}$. By Theorem 2.3, finding the second eigenvector of the Laplacian thus allows us to find a partition of ratio $O(\sqrt{g/n})$ for a graph of bounded degree. There is no guarantee that this partition has a similar number of vertices in each of the two sets. However, a theorem of Lipton and Tarjan [19] implies that a simple method based on repeated application of this algorithm can be used to give a bisection of size $O(\sqrt{gn})$.

For every g , Gilbert, Hutchinson, and Tarjan exhibited a class of bounded degree graphs that have no bisections smaller than $O(\sqrt{gn})$ [14]. This implies that our algorithm gives the best results possible, in general. Furthermore, it establishes the asymptotic tightness of our eigenvalue bound, as a smaller bound would show that every genus g graph has a partition of size $o(\sqrt{gn})$.

Putting all of this together yields our main algorithmic result, the following.

THEOREM 2.5. *Let G be a genus g graph of bounded maximum degree. There is a polynomial time algorithm that produces cuts of ratio $O(\sqrt{g/n})$ and vertex bisections of size $O(\sqrt{gn})$ in G , and both of these values are optimal.*

All that remains of the proof of Theorem 2.5 is the eigenvalue bound set forth in Theorem 2.3, which is the goal of the remainder of this paper.

3. Outline of the proof of Theorem 2.3. The proof of Theorem 2.3 necessitates the introduction of a good deal of technical machinery. Before launching into several pages of definitions and background theorems, we feel that a brief roadmap of where we're going will be helpful.

The basic motivation for our approach comes from an observation made by Spielman and Teng [25]. They noted that one can obtain bounds on the eigenvalues of a graph G from a nice representation of G on the unit sphere in \mathbb{R}^3 , known as a *circle packing* for G . This is a presentation of the graph on the sphere so that the vertices are the centers of a collection of circles, and the edges between vertices correspond to tangencies of their respective circles, as shown in Figure 4.1. Only planar graphs can be embedded as such if we require that the circles have disjoint interiors. However, if we allow the circles to overlap, as shown in Figure 4.2, we can represent nonplanar graphs as well. This will give rise to a weaker bound in which the eigenvalue bound is multiplied by the maximum number of circles containing a given point (i.e., the number of layers of circles on the sphere).

There is a well developed theory of circle packings, both on the sphere and on higher genus surfaces. The portions of it that we shall use will tell us two main things:

1. We can realize our graph as a circle packing of circles with disjoint interiors on some genus g surface.
2. The theory of discrete circle packings can be thought of as a discrete analogue of classical complex function theory, and many of the results of the latter carry

over to the former.

In classical complex analysis, you can put a complex analytic structure on a genus g surface to obtain a Riemann surface. Any genus g Riemann surface has a map to the sphere that is almost everywhere k -to-one for $k = O(g)$, with only $O(g)$ bad points at which this fails. With this as motivation, we shall try to use the representation of G as a circle packing on a genus g surface to obtain a representation of it as a circle packing on the sphere with $O(g)$ layers.

Unfortunately, the discrete theory is more rigid than the continuous one, and finding such a representation will turn out to be impossible. Instead, we shall actually pass to the continuous theory to prove our result. To do this, we shall provide a subdivision lemma that shows that it suffices to prove Theorem 2.3 for graphs that have circle packings with very small circles. We shall then show that the smooth map that we have from the Riemann surface to the sphere will take *almost* all of the circles of our circle packing to curves on the sphere that are *almost* circles. We will then show that this representation of our graph as an approximate circle packing is enough to provide our desired bounds.

4. Introduction to circle packings. Our proof of Theorem 2.3 operates by obtaining a nice geometric realization of G . We obtain this realization using the theory of circle packings. In this section, we shall review the basics of circle packing theory and quote the main results that our proof will employ. For a more comprehensive treatment of this theory and a historical account of its origins, see [26].

Loosely speaking, a circle packing is a collection of circles on a surface with a given pattern of tangencies. We remark at the outset that the theory that we are discussing *is not the same* as the classical theory of sphere packing. Our theory is concerned with the combinatorics of the tangency patterns, *not* with the maximum number of circles that one can fit in a small region. The coincidence of nomenclature is just an unfortunate historical accident.

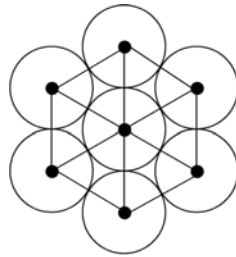


FIG. 4.1. A univalent circle packing with its associated graph.

4.1. Planar circle packings. For simplicity, we begin by discussing circle packings in the plane.

DEFINITION 4.1. A planar circle packing \mathcal{P} is a finite collection of (possibly overlapping) circles C_1, \dots, C_n of respective radii r_1, \dots, r_n in the complex plane \mathbb{C} . If all of the C_i have disjoint interiors, we say that \mathcal{P} is univalent.

The associated graph $A(\mathcal{P})$ of \mathcal{P} is the graph obtained by assigning a vertex v_i to each circle C_i and connecting v_i and v_j by an edge if and only if C_i and C_j are mutually tangent.

This is illustrated in Figures 4.1 and 4.2.

We thus associate a graph with every circle packing. It is clear that every graph

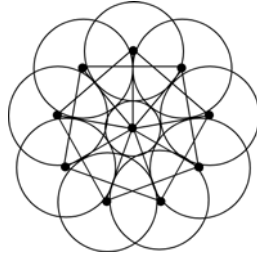


FIG. 4.2. A nonunivalent circle packing with its associated graph.

associated with a univalent planar circle packing is planar. A natural question to ask is whether every planar graph can be realized as the associated graph of some planar circle packing. This is answered in the affirmative by the Koebe–Andreev–Thurston theorem [18, 1, 28].

THEOREM 4.2 (Koebe–Andreev–Thurston). *Let G be a planar graph. There exists a planar circle packing \mathcal{P} such that $A(\mathcal{P}) = G$.*

This theorem also contains a uniqueness result, but we have not yet developed the machinery to state it. We shall generalize this theorem in section 4.3, at which point we shall have the proper terminology to state the uniqueness part of the theorem.

We note that if we map the plane onto the sphere by stereographic projection, circles in the plane will be sent to circles on the sphere, so this theorem can be interpreted as saying that every genus 0 graph can be represented as a circle packing on the surface of a genus 0 surface. This suggests that we attempt to generalize this theorem to surfaces of higher genus. The theory of circle packings on surfaces of arbitrary genus acts in many ways like a discrete analogue of classical Riemann surface theory. As such, a basic background in Riemann surfaces is necessary to state or motivate many of its results. It is to this that we devote the next section.

4.2. A very brief introduction to Riemann surface theory. In this section, we provide an informal introduction to Riemann surface theory. Our goal is to provide geometric intuition, not mathematical rigor. We assume some familiarity with the basic concept of a manifold, as well as with the basic definitions of complex analysis. For a more complete exposition of the theory, see [13].

We recall that an n -dimensional manifold is a structure that looks locally like \mathbb{R}^n . More formally, we write our manifold M as a topological union of open sets S_i , each endowed with a homeomorphism $\varphi_i : S_i \rightarrow B_n$, where B_n is the ball $\{|x| < 1 \mid x \in \mathbb{R}^n\}$. Furthermore, we require a compatibility among these maps to avoid cusps and such. To this end, we mandate that the compositions $\varphi_j \circ \varphi_i^{-1} : \varphi_i(S_i \cap S_j) \rightarrow \varphi_j(S_i \cap S_j)$ be diffeomorphisms. The orientable 2-dimensional manifolds are precisely the genus g surfaces described above.

An n -dimensional complex manifold is the natural complex analytic generalization of this. We write our manifold M as a union of open sets S_i and endow each such set with a homeomorphism $\varphi_i : S_i \rightarrow B_{\mathbb{C}^n}$, where $B_{\mathbb{C}^n}$ is the complex unit ball $\{|x| < 1 \mid x \in \mathbb{C}^n\}$. Now, instead of requiring that the compositions of these functions obey a smooth compatibility condition, we require that they obey an analytic one: we demand that the compositions $\varphi_i \circ \varphi_j^{-1}$ be biholomorphic maps.

As such, an n -dimensional complex manifold M is a $2n$ -dimensional real manifold with additional complex analytic structure. This structure allows us to transfer over many of the definitions from standard complex analysis. The basic idea is that we

define these notions as before on the S_i , and the compatibility condition allows them to make sense as global definitions. In particular, if $M = (S_i^M, \phi_i^M)$ and $N = (S_j^N, \phi_j^N)$ are complex manifolds of the same dimension, we say that a function $f : M \rightarrow N$ is holomorphic if its restriction to a map $f_{ij} : S_i^M \rightarrow S_j^N$ is holomorphic for all i and j . Since the compositions $\varphi_i^M \circ (\varphi_j^M)^{-1}$ and $\varphi_i^N \circ (\varphi_j^N)^{-1}$ are holomorphic, this notion makes sense where the regions overlap.

DEFINITION 4.3. *A Riemann surface is a 1-dimensional complex manifold.*

In this paper, we shall take all of our Riemann surfaces to be compact. Since there is a natural way to orient the complex plane, we note that the complex structure can be used to define an orientation on the manifold. As such, all complex manifolds, and in particular all Riemann surfaces, are orientable. Compact Riemann surfaces are thus, topologically, 2-dimensional orientable real manifolds. Every compact Riemann surface is therefore topologically one of the genus g surfaces discussed above. The complex structure imposed by the φ_i , however, varies much more widely, and there are many different such structures that have the same underlying topological space.

Nothing in the definition of a Riemann surface supplies a metric on the surface. Indeed, there is no requirement that the different ϕ_i agree in any way about the distance between two points in their intersection. One can assign many different metrics to the surface. However, it turns out that there is way to single out a unique metric on the surface, called the *metric of constant curvature*. This allows us to supply an intrinsic notion of distance on any Riemann surface. In particular, this allows us to define a circle on our Riemann surface to be a simple closed curve that is contractible on the surface and all of whose points lie at a fixed distance from some center.

One particularly important Riemann surface that we shall consider is the Riemann sphere, which we denote $\widehat{\mathbb{C}}$. It is topologically a sphere. It should be thought of as being obtained by taking the complex plane and adjoining a single point called ∞ . One way of visualizing its relation to \mathbb{C} is to consider the stereographic projection away from the North Pole of a sphere onto a plane. The North Pole corresponds to ∞ , and the rest of the sphere corresponds to \mathbb{C} .

We recall from single variable complex analysis that the requirement that a map be analytic is quite a stringent one, and that it imposes a significant amount of local structure on the map. Let $f : \mathbb{C} \rightarrow \mathbb{C}$ be nonconstant and analytic in a neighborhood of the origin, and assume without loss of generality that $f(0) = 0$. There is some neighborhood of the origin in which f can be expressed as a power series $f(z) = a_1z + a_2z^2 + a_3z^3 + \dots$. If $a_1 \neq 0$, $f(z)$ is analytically invertible in some neighborhood of the origin, and thus it is locally an isomorphism. In particular, it is *conformal*—it preserves the angles between intersecting curves, and the image of an infinitesimal circle is another infinitesimal circle.

If $a_1 = 0$ and a_n is the first nonzero coefficient in its power series, f has a *branch point of order n* at the origin. In this case, f operates, up to a scale factor and lower order terms, like the function $f(z) = z^n$. This function is n -to-one on a small neighborhood of the origin, excluding the origin itself. It sends only 0 to 0, however. The preimages of the points in this small neighborhood thus trace out n different “sheets” that all intersect at 0. This confluence of sheets is the only sort of singularity that can appear in an analytic map. We note that the angles between curves intersecting at the branch point are not preserved, but they are instead divided by n .

This local behavior is identical for Riemann surfaces. From this, we can deduce that if $f : M \rightarrow N$ is an analytic map of Riemann surfaces, it has some well-defined

degree k . For all but finitely many points p in N , $\#f^{-1}(p) = k$. The preimage of each of these points looks like a collection of k sheets, and f has nonzero derivative at all of them. There exist some points $q \in M$ at which $f'(q) = 0$. At each such point there is a branch point, so the sheets intersect, and $f(q)$ has fewer than k preimages.

However, the global structure of Riemann surfaces provides further constraints on maps between them, and there are, generally speaking, very few functions $f : M \rightarrow N$ of a given degree. For example, topological arguments, using the local form of analytic maps described above, show that there are no degree 1 maps from the torus to the sphere, and no degree 2 maps from the genus 2 surface to the sphere.

There is a deep theory of maps of Riemann surfaces that describes rather precisely when a map of a given degree exists between two Riemann surfaces, and, if it exists, where and how such a map must branch. Of this theory we shall require only one main result, which is a direct corollary of the celebrated Riemann–Roch theorem.

THEOREM 4.4. *Let M be a Riemann surface of genus g . There exists an analytic map $f : M \rightarrow \widehat{\mathbb{C}}$ of degree $O(g)$ and with $O(g)$ branch points.*

4.3. Circle packings on surfaces of arbitrary genus. We now have the machinery in place to deal with general circle packings. Throughout this section, let G be a graph of genus g , and suppose that it is embedded on a genus g surface S so that none of its edges cross. The graph G divides S into faces. We say that G is a *fully triangulated* graph if all of these faces are triangles, in which case we say that it gives a *triangulation* of S . If G is not fully triangulated, one can clearly add edges to it to make it so. It will follow immediately from (5.2) in section 5 that this will only increase $\lambda_2(G)$, so we shall assume for convenience that G gives a triangulation of S . We are now ready to define our primary objects of study, as follows.

DEFINITION 4.5. *Let S be a compact Riemann surface endowed with its metric of constant curvature. A circle packing \mathcal{P} on S is a finite collection of (possibly overlapping) circles C_1, \dots, C_n of respective radii r_1, \dots, r_n on the surface of S . If all of the C_i have disjoint interiors, we say that \mathcal{P} is univalent.*

The associated graph $A(\mathcal{P})$ of \mathcal{P} is the graph obtained by assigning a vertex v_i to each circle C_i and connecting v_i and v_j by an edge if and only if C_i and C_j are mutually tangent. Alternatively, we say that \mathcal{P} is a circle packing for $A(\mathcal{P})$ on S .

The main result on circle packings that we shall use is the circle packing theorem, which is the natural extension of the Koebe–Andreiev–Thurston theorem to this more general setting. It was originally proven in a restricted form by Beardon and Stephenson [4] and then proven in full generality by He and Schramm [17].

THEOREM 4.6 (circle packing theorem). *Let G be a triangulation of a surface of genus g . There exists a Riemann surface S of genus g and a univalent circle packing \mathcal{P} such that \mathcal{P} is a circle packing for G on S . This packing is unique up to automorphisms of S .*

If G is embedded in a surface of genus g but is not fully triangulated, the Riemann surface and circle packing guaranteed by the theorem still exist, but they need not be unique.

The complex structure on the Riemann surface allows us to define the angle at which two edges of a face meet. If the points u , v , and w are the vertices of a face, we denote the angle between the edges \overline{uv} and \overline{vw} at v by $\langle uvw \rangle$. We can thus define the angle sum at a vertex to be $\sum \langle uvw \rangle$, where the sum is taken over all faces containing v . If \mathcal{P} is a univalent sphere packing, the angle sum at any vertex of $A(\mathcal{P})$ is clearly 2π .

In a nonunivalent circle packing, it is possible for the circles at a point to wrap around the point more than once. In the case of a nonunivalent circle packing, the

edges of its associated graph may intersect, but we can still define an associated triangulation of the surface—there just may be more than one triangle covering a given point. We can therefore compute the angle sum at a point. In this case, it need not be 2π . However, the circles must wrap around the vertex an integral number of times, so it must be some multiple $2\pi k$. (See Figure 4.2.) We then say that the vertex is a *discrete branch point of order k* .

These discrete branch points behave very much like the continuous branch points present on Riemann surfaces. In fact, there is an extensive theory that shows that a large portion of the theory of Riemann surfaces has an analogue in the discrete realm of circle packing. One can define maps of circle packings, just as one can define maps of Riemann surfaces. They consist of a correspondence of the circles on one surface to those on another in a way that commutes with tangency. While analytic maps send infinitesimal circles to infinitesimal circles, maps of circle packings send finite circles to finite circles. The analogue of branched covering maps in Riemannian geometry takes univalent circle packings and places them as nonunivalent circle packings on other surfaces. Unfortunately, these maps are somewhat rarer than their continuous analogues.

In particular, if we have a circle packing on a genus g surface S , there is no known analogue of the Riemann–Roch theorem, and thus no analogue of Theorem 4.4. We are therefore not guaranteed that there is a nonunivalent circle packing on the sphere carrying the same associated graph. Intuitively, this comes from the fact that the analytic maps from S to $\widehat{\mathbb{C}}$ are required to be branched over a very restricted locus of points. The discrete maps, however, can only be branched over the centers of circles. If there does not exist an admissible set of branch points among the centers of the circles, we will have difficulty constructing a discrete analytic map. This will lie at the root of many of the technical difficulties that we shall face in the remainder of this paper.

5. An eigenvalue bound. In this section, we prove Theorem 2.3. The proof will assume a technical lemma whose proof we shall postpone until section 6.

We begin by recalling the expression of the Fiedler value of G as a so-called *Rayleigh quotient*:

$$(5.1) \quad \lambda_2 = \min_{x \perp (1, \dots, 1)^T} \frac{x^T L(G)x}{x^T x}.$$

A straightforward calculation shows that for $x = (x_1, \dots, x_n)^T \in \mathbb{R}^n$,

$$x^T L(G)x = \sum_{(i,j) \in E} (x_i - x_j)^2,$$

so that (5.1) becomes

$$(5.2) \quad \lambda_2 = \min_{x \perp (1, \dots, 1)^T} \frac{\sum_{(i,j) \in E} (x_i - x_j)^2}{x^T x}.$$

As noted by Spielman and Teng [25], it follows easily from (5.2) that we can replace the scalar values x_i with vectors $v_i \in \mathbb{R}^k$, so that

$$(5.3) \quad \lambda_2 = \min \frac{\sum_{(i,j) \in E} \|v_i - v_j\|^2}{\sum_{i=1}^n \|v_i\|^2},$$

where the minimum is taken over all sets of n -vectors such that $\sum v_i = (0, \dots, 0)^T$ and such that at least one of the v_i is nonzero.

The general goal is thus to find a set of v_i that gives a small value for this quotient. The v_i that we use will almost be the centers of a nonunivalent circle packing on the unit sphere $S^2 \subseteq \mathbb{R}^3$. The efficacy of this follows from the following theorem, which follows easily from the work of Spielman and Teng [25].

THEOREM 5.1. *Let \mathcal{P} be a circle packing on the sphere $S^2 = \{x \in \mathbb{R}^3 \mid \|x\|^2 = 1\}$ so that the graph $A(\mathcal{P})$ has no vertex of degree greater than Δ . Suppose further that the packing is of degree k , so that no point on the sphere is contained in the interior of more than k circles, and that the centroid of the centers of the circles is the origin. Then the Fiedler value*

$$\lambda_2(A(\mathcal{P})) \leq O\left(\frac{\Delta k}{n}\right).$$

Proof. The proof follows from (5.3). Let the circles be C_1, \dots, C_n , and let the corresponding radii be r_1, \dots, r_n . Let $v_i \in \mathbb{R}^3$ be the x, y , and z coordinates of the center of the i th circle. The sum $\sum v_i = 0$ by assumption, so λ_2 is less than or equal to the fraction in (5.3). Since all of the v_i are on the unit sphere, we have $\sum \|v_i\|^2 = n$, so it just remains to bound the numerator. If there is an edge (i, j) , the two circles C_i and C_j must be mutually tangent, so that $\|v_i - v_j\|^2 \leq (r_i + r_j)^2 \leq 2(r_i^2 + r_j^2)$. It thus follows that

$$\sum_{(i,j) \in E} \|v_i - v_j\|^2 \leq \sum_{(i,j) \in E} 2(r_i^2 + r_j^2) \leq 2\Delta \sum_{i=1}^n r_i^2.$$

However, the total area of all of the circles is less than or equal to k times the area of the sphere, since the circle packing is of degree k . We thus have that $\sum_{i=1}^n r_i^2 \leq O(k)$, from which the desired result follows. \square

This suggests that we use the circle packing theorem (Theorem 4.6) to embed our graph on a genus g surface and then try to use some analogue of Theorem 4.4 to obtain a branched circle packing on the sphere of degree $O(g)$. Unfortunately, as previously noted, such a circle packing need not exist, due to the restrictiveness of the discrete theory. Thus, we shall instead show that a certain subdivision process on our graph does not significantly decrease $n\lambda_2$. We shall then show that performing this subdivision enough times causes our discrete circle packing to approximate a continuous structure on the Riemann surface, at which point we can use the continuous theory in addition to the discrete one.

The refinement procedure that we shall use is called “hexagonal refinement.” It operates on a triangulation of a surface by replacing each triangle with four smaller triangles, as shown in Figure 5.1. This process produces another triangulation of the same surface, so we can iterate it arbitrarily many times.

LEMMA 5.2. *Let G be a graph with n vertices, m edges, and maximum degree Δ that triangulates some surface without boundary, and let G' be the graph with n' vertices and m' edges obtained by performing k successive hexagonal refinements on G . Then*

$$n\lambda_2(G) \leq C(\Delta)n'\lambda_2(G').$$

For the sake of continuity, we defer this proof to section 6.

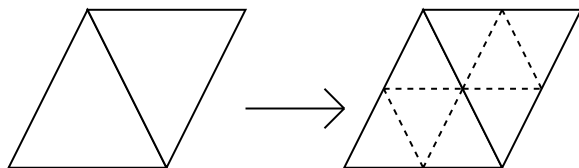


FIG. 5.1. The hexagonal subdivision procedure applied to a triangulation with two triangles.

The refinement process replaces each triangle in our graph with four smaller triangles. If all of the original triangles remained the same size and shape, this would imply that performing enough hexagonal refinements would give rise to a circle packing whose circles have arbitrarily small radii. However, it is possible for the original triangles to change size and shape as we refine, so this is no longer obvious. Nevertheless, it remains true, as shown by the following lemma.

LEMMA 5.3. *Let G be a graph that triangulates a genus g Riemann surface without boundary, and let $G^{(k)}$ be the graph obtained by performing k hexagonal refinements on G . For every $\epsilon > 0$, there exists some k_ϵ so that for all $\ell \geq k_\epsilon$ every circle in $G^{(\ell)}$ has radius less than ϵ .*

Proof. This was essentially proven by Rodin and Sullivan [23]; their proof, however, was stated for only the genus 0 case. The precise statement above was proven by Bowers and Stephenson [5]. \square

We get a new Riemann surface for each iteration of the refinement procedure. It is intuitive that, as the number of iterations grows and the circles in the refined graph get arbitrarily small, the Riemann surfaces will somehow converge, and the embedding of the graph on these Riemann surfaces will somehow stabilize. This can be made formal by the following lemma.

LEMMA 5.4. *Let G be a graph that triangulates a genus g compact Riemann surface without boundary, let $G^{(k)}$ be the result of performing k hexagonal refinements on G , and let $S^{(k)}$ be the Riemann surface on which $G^{(k)}$ is realized as a circle packing. Further, let $h_k : S^{(k)} \rightarrow S^{(k+1)}$ be the map that takes a triangle to its image under the subdivision procedure by the obvious piecewise-linear map. The sequence of surfaces $\{S^{(k)}\}$ converges in the moduli space of genus g surfaces, and the sequence of maps $\{h_k\}$ converges to the identity.*

Proof. This has been shown by Bowers and Stephenson [5]. \square

We shall also require one last definition, as follows.

DEFINITION 5.5. *Let $f : X \rightarrow Y$ be a map between two locally Euclidean metric spaces. The quantity*

$$H_f(x, r) = \frac{\max_{|x-y|=r} |f(x) - f(y)|}{\min_{|x-y|=r} |f(x) - f(y)|} - 1$$

is called the radius r distortion of f at x .

We are now finally ready to prove Theorem 2.3.

Proof of Theorem 2.3. Using the circle packing theorem (Theorem 4.6), realize the graph $G = G^{(0)}$ as a circle packing on some Riemann surface S of genus g . Let $G^{(k)}$ be the result of performing k hexagonal refinements on G , and let $S^{(k)}$ be the Riemann surface on which it can be realized as a circle packing. By Theorem 4.4, there exists an analytic map $f^{(k)}$ from $S^{(k)}$ to the Riemann sphere of degree $O(g)$

and with $O(g)$ branch points. Embed the Riemann sphere as the unit sphere in \mathbb{R}^3 using the conformal map given by inverse stereographic projection. By the work of Spielman and Teng (Theorem 9 of [25]), postcomposing with a Möbius transformation allows us to assume, without loss of generality, that the centroid of the images of the vertices of each $G^{(k)}$ under $f^{(k)}$ is the origin. By Lemma 5.4, the $S^{(k)}$ converge to some surface $S^{(\infty)}$, and the $f^{(k)}$ can be chosen so as to converge to some continuous limit map $f^{(\infty)}$.

By Lemma 5.2, it suffices to prove the theorem for an arbitrarily fine hexagonal refinement of the original graph. Away from its branch points, a map of Riemann surfaces is conformal, meaning it sends infinitesimal circles to infinitesimal circles. In particular, given a map $f : S \rightarrow \widehat{\mathbb{C}}$, the compactness of S guarantees that for every $\epsilon, \kappa > 0$ there exists a $\delta > 0$ so that the radius δ' distortion $H_f(x, \delta')$ is less than ϵ for every x that is at least distance κ from any branch point and any $\delta' \leq \delta$. In fact, by the convergence results of the last paragraph, there exist some N and δ such that this holds for every $f^{(k)}$ with $k > N$. Fix ϵ and κ , and let δ and N be chosen so that this is true. By possibly increasing N if necessary, we can assume by Lemma 5.3 that all of the circles on $S^{(k)}$ have radius at most δ for all $k > N$.

Let k be at least N . We shall break $S^{(k)}$ into two parts, $S^{(k)} = S_1^{(k)} \cup S_2^{(k)}$, as follows. Construct a ball of radius κ around each branch point of $f^{(k)}$, and let $S_2^{(k)}$ be the union of these balls. Let $S_1^{(k)}$ be the complement $S^{(k)} \setminus S_2^{(k)}$.

We can now use (5.3) to bound λ_2 , just as in the proof of Theorem 5.1. Let $G^{(k)}$ have n_k vertices. The denominator of (5.3) is equal to n_k , so it suffices to bound the numerator. We shall separately consider the circles contained entirely in $S_1^{(k)}$ and those that intersect $S_2^{(k)}$.

We begin with the circles contained in $S_1^{(k)}$. Every circle of the packing gets mapped by f to some connected region on $\widehat{\mathbb{C}}$, and there are at most $O(g)$ such regions covering any point of the sphere. Let C be a circle in $S_1^{(k)}$, let D be the diameter function, which takes a region to the length of the longest geodesic it contains, and let A be the area function. Since the radius δ distortion of f inside of $S_1^{(k)}$ is at most ϵ , and the radius of C is at most δ , the ratio $D^2(f(C))/A(f(C))$ is at most $O(1 + \epsilon)$. Using the same argument as in the proof of Theorem 5.1, the vertex at the center of a circle C cannot contribute more than $O(dD^2(f(C)))$ to the sum, and the total area of the regions from $S_1^{(k)}$ cannot exceed $O(g)$, so the total contribution to the numerator of the vertices in $S_1^{(k)}$ cannot be more than $O(dg(1 + \epsilon))$.

If this were the only term in the numerator, we could complete the proof by setting ϵ to be a constant. It thus remains to show that the contribution from the circles intersecting $S_2^{(k)}$ can be made small. To do this, we need only show that the contribution $\theta^{(k)}(x)$ to the numerator per unit area at a point x from these circles remains bounded as we subdivide, since we can make the area of $S_2^{(k)}$ arbitrarily small by sending κ to zero, and thus the area of the circles intersecting $S_2^{(k)}$ will go to zero as k goes to infinity and the circles get arbitrarily small. Our argument is similar to one used by McCaughan to analyze the recurrence of random walks on circle packings [20].

Let $x_i, i = 1, \dots, 3$, be the coordinate functions on \mathbb{R}^3 , and let $f^{(k)*}x_i$ be their pullbacks along $f^{(k)}$ to $S^{(k)}$. (That is, if y is a point on $S^{(k)}$, $f^{(k)*}x_i(y) = x_i(f^{(k)}(y))$.) In addition, let $C_1^{(k)}$ and $C_2^{(k)}$ be a pair of adjacent circles in $S_2^{(k)}$ with respective radii $r_1^{(k)}$ and $r_2^{(k)}$ and respective centers $c_1^{(k)}$ and $c_2^{(k)}$. The contribution of the correspond-

ing edge in $G^{(k)}$ to the numerator of (5.3) will be

$$(5.4) \quad \left\| \left(f^{(k)*} x_i(c_1^{(k)}) \right)_{i=1}^3 - \left(f^{(k)*} x_i(c_2^{(k)}) \right)_{i=1}^3 \right\|^2 = \sum_{i=1}^3 \left(f^{(k)*} x_i(c_1^{(k)}) - f^{(k)*} x_i(c_2^{(k)}) \right)^2.$$

The distance between $c_1^{(k)}$ and $c_2^{(k)}$ equals $r_1^{(k)} + r_2^{(k)}$. As k goes to infinity, the radii $r_1^{(k)}$ and $r_2^{(k)}$ both go to zero, by Lemma 5.3. By the smoothness of the $f^{(k)}$, their convergence to $f^{(\infty)}$, and the compactness of their domains, we can approximate each term on the right-hand side of (5.4) arbitrarily well by its first order approximation, so that

$$(5.5) \quad \left(f^{(k)*} x_i(c_1^{(k)}) - f^{(k)*} x_i(c_2^{(k)}) \right)^2 \leq (1 + o(1))(r_1^{(k)} + r_2^{(k)})^2 \|\nabla f^{(k)*} x_i(c_1^{(k)})\|^2$$

as k goes to infinity and the distance between $c_1^{(k)}$ and $c_2^{(k)}$ shrinks to zero.

The right-hand side of (5.5) is bounded above by

$$(5.6) \quad (2 + o(1))[(r_1^{(k)})^2 + (r_2^{(k)})^2] \|\nabla f^{(k)*} x_i(c_1^{(k)})\|^2 = O(1)[(r_1^{(k)})^2 \|\nabla f^{(k)*} x_i(c_1^{(k)})\|^2 + (r_2^{(k)})^2 \|\nabla f^{(k)*} x_i(c_2^{(k)})\|^2].$$

The degree of our graph is bounded, so every vertex appears in at most a constant number of edges. If we sum the right-hand side of (5.6) over all of the edges in our graph, the total contribution of terms involving a fixed circle of radius r centered at c is thus bounded above by

$$O(1)r^2 \|\nabla f^{(k)*} x_i(c)\|^2,$$

so the contribution per unit area is bounded above by

$$O(1) \|\nabla f^{(k)*} x_i(c)\|^2.$$

This clearly remains bounded as k goes to infinity and $f^{(k)}$ approaches $f^{(\infty)}$. It thus follows that the contribution to the numerator of (5.3) of the vertices in $S_2^{(k)}$ tends to zero as k goes to infinity and κ is made arbitrarily small. By setting ϵ to be a constant and sending κ to zero, Theorem 2.3 follows. \square

6. The proof of Lemma 5.2. In this section, we shall prove Lemma 5.2. In proving this bound, it will be convenient to consider the following weighted form of the Laplacian.

DEFINITION 6.1. *The weighted Laplacian $\mathcal{L}^W(G)$ of a graph G is the matrix*

$$\mathcal{L}^W(G) = W^{-1/2}L(G)W^{-1/2},$$

where $L(G)$ is the Laplacian of G , and W is a diagonal matrix whose i th diagonal entry w_i is strictly positive for all i .

We shall denote the eigenvalues of $\mathcal{L}^W(G)$ by $\tilde{\lambda}_1^W(G) \leq \dots \leq \tilde{\lambda}_n^W(G)$ and the corresponding eigenvectors by $\tilde{v}_1^W(G) \dots \tilde{v}_n^W(G)$. A straightforward calculation shows

that the weighted Laplacian has $\tilde{\lambda}_1^W = 0$ and $\tilde{v}_1^W = W^{1/2}\mathbf{1}$. Our main quantity of interest will be $\tilde{\lambda}_2^W(G)$, which we can compute using a weighted analogue of the Rayleigh quotient:

$$(6.1) \quad \tilde{\lambda}_2^W = \min_{x \perp W\mathbf{1}} \frac{\sum_{(i,j) \in E} (x_i - x_j)^2}{\sum_i x_i^2 w_i}.$$

The second eigenvector $\tilde{v}_2^W(G)$ equals $W^{1/2}x$, where x is the vector that achieves the minimum in (6.1).

If all of the weights are $\Theta(1)$, standard linear algebra shows that $\lambda_2(G)$ and $\tilde{\lambda}_2^W(G)$ differ by at most a constant factor, so proving a bound on one implies a bound on the other. (See Chung’s book [9] for detailed proofs of the above facts and for other foundational information about the weighted Laplacian.)

Before we can proceed to the body of the proof of Lemma 5.2, we require two fairly general technical lemmas about independent random variables.

LEMMA 6.2. *Let a_1, \dots, a_n be independent real-valued random variables, possibly drawn from different probability distributions. Let $w_1, \dots, w_n \in \mathbb{R}^+$ be strictly positive constants. If the expectation $\mathbb{E}[\sum_i w_i a_i] = 0$, then*

$$\mathbb{E} \left[\left(\sum_j w_j a_j \right)^2 \right] \leq \mathbb{E} \left[\sum_j w_j^2 a_j^2 \right].$$

Proof. This follows by expanding the left-hand side:

$$\begin{aligned} \mathbb{E} \left[\left(\sum_j w_j a_j \right)^2 \right] &= \mathbb{E} \left[\sum_i w_i^2 a_i^2 \right] + \mathbb{E} \left[\sum_i w_i a_i \left(\sum_{j \neq i} w_j a_j \right) \right] \\ &= \mathbb{E} \left[\sum_i w_i^2 a_i^2 \right] + \sum_i -(\mathbb{E}[w_i a_i])^2 \\ &\leq \mathbb{E} \left[\sum_j w_j^2 a_j^2 \right], \end{aligned}$$

where the second equality follows from the independence of the variables and the fact that the sum of their expectations is zero. \square

We shall now use this lemma to establish our second lemma, which is the one that will actually appear in our main proof.

LEMMA 6.3. *Let a_1, \dots, a_n be independent real-valued random variables, possibly drawn from different probability distributions, and let $w_1, \dots, w_n \in \mathbb{R}^+$ be strictly positive constants such that $\mathbb{E}[\sum_i w_i a_i] = 0$. Let $a = (a_1, \dots, a_n)$, and let $w_{\max} = \max_i w_i$. Further let*

$$b = \left(\frac{1}{\sum_i w_i} \sum_i w_i a_i \right) \mathbf{1} \quad \text{and} \quad c = a - b.$$

Then

$$\mathbb{E} \left[\sum_i w_i c_i^2 \right] \geq \left(1 - \frac{w_{\max}}{\sum_i w_i} \right) \mathbb{E} \left[\sum_i w_i a_i^2 \right].$$

Proof. This follows by direct calculation:

$$\begin{aligned}
\mathbb{E} \left[\sum_i w_i c_i^2 \right] &= \mathbb{E} \left[\sum_i w_i \left(a_i - \frac{1}{\sum_j w_j} \left(\sum_j w_j a_j \right) \right)^2 \right] \\
&= \mathbb{E} \left[\sum_i w_i a_i^2 \right] + \frac{1}{(\sum_i w_i)^2} \mathbb{E} \left[\sum_i w_i \left(\sum_j w_j a_j \right)^2 \right] \\
&\quad - \frac{2}{\sum_i w_i} \mathbb{E} \left[\sum_i w_i a_i \left(\sum_j w_j a_j \right) \right] \\
&= \mathbb{E} \left[\sum_i w_i a_i^2 \right] + \frac{1}{\sum_i w_i} \mathbb{E} \left[\left(\sum_j w_j a_j \right)^2 \right] - \frac{2}{\sum_i w_i} \mathbb{E} \left[\left(\sum_j w_j a_j \right)^2 \right] \\
&= \mathbb{E} \left[\sum_i w_i a_i^2 \right] - \frac{1}{\sum_i w_i} \mathbb{E} \left[\left(\sum_j w_j a_j \right)^2 \right] \\
&\geq \mathbb{E} \left[\sum_i w_i a_i^2 \right] - \frac{1}{\sum_i w_i} \mathbb{E} \left[\sum_j w_j^2 a_j^2 \right] \\
&= \mathbb{E} \left[\sum_i \left(1 - \frac{w_i}{\sum_j w_j} \right) w_i a_i^2 \right] \\
&\geq \left(1 - \frac{w_{\max}}{\sum_i w_i} \right) \mathbb{E} \left[\sum_i w_i a_i^2 \right],
\end{aligned}$$

where second-to-last inequality follows from Lemma 6.2. \square

We are now prepared to prove Lemma 5.2.

Proof of Lemma 5.2. Let $G = (V_G, E_G)$ be the original graph, and let $G' = (V_{G'}, E_{G'})$ be the graph that results from performing k successive hexagonal refinements on G . The embeddings into surfaces endow both G and G' with triangulations; let T_G and $T_{G'}$ be the respective sets of triangles in these triangulations. There is a natural inclusion $\iota : V_G \hookrightarrow V_{G'}$, since the subdivision procedure only adds vertices to the original set. There is also a map $\eta : T_{G'} \rightarrow T_G$ that takes a triangle from the subdivided graph to the one in the original graph from which it arose. For a vertex v in either graph, let $N(v)$ be the set of triangles containing it. For a vertex $w \in V_G$, let $P(w) = \eta^{-1}(N(w))$ be the set of triangles in $T(G')$ taken by η to elements of $N(w)$. (See Figure 6.1.)

Our proof will proceed by producing a randomized construction of a subgraph H of G' . Given a vector that assigns a value to every vertex of G' , we can obtain such a vector on H by restriction. We shall also show how to use such a vector on H to construct such a vector on G . The vectors on the different graphs will give rise to Rayleigh quotients on the graphs (some of which will be weighted), where the Rayleigh quotients for G and H will depend on the random choices made in the construction of H . By relating the terms in the different Rayleigh quotients, we shall then provide a probabilistic proof that there exists an H that gives rise to a small Rayleigh quotient

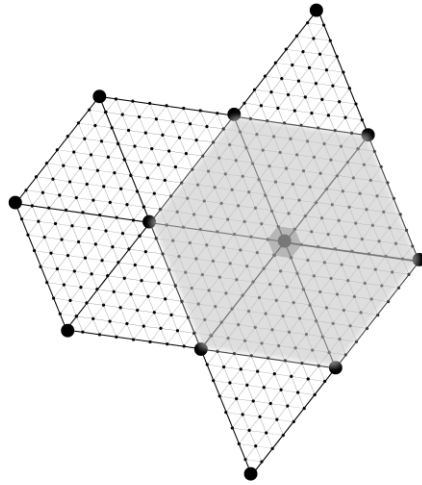


FIG. 6.1. A subdivided graph, with $P(w)$ and $N(w)$ shaded for a vertex w .

on G , which will suffice to prove our desired bound.

H will be produced by randomly choosing a representative in $V_{G'}$ for each vertex in V_G and representing every edge in E_G by a randomly chosen path in G' between the representatives of its endpoints.

We first construct the map $\pi_V : V_G \rightarrow V_{G'}$ that chooses the representatives of the vertices. For each $v \in V_G$ we choose $\pi_V(v)$ uniformly at random from the vertices contained in $P(v)$ that are at least as close to $\iota(v)$ as to $\iota(w)$ for any other $w \in V_G$. Vertices in $P(v)$ that are equally close to $\iota(v)$ and $\iota(w)$ should be arbitrarily assigned to either v or w , but not both.

We now construct π_E , which maps edges in E_G to paths in G' . Let $e = (v_1, v_2)$ be an edge in G , and let w_1 and w_2 equal $\pi_V(v_1)$ and $\pi_V(v_2)$, respectively. The two neighborhoods in G , $N(v_1)$ and $N(v_2)$, share exactly two triangles, t_1 and t_2 . Let x be a vertex randomly chosen from the vertices in $\eta^{-1}(t_1 \cup t_2)$. We shall construct a path from each w_i ($i = 1, 2$) to x , so that their composition gives a path from w_1 to w_2 . We shall use the same construction for each, so, without loss of generality, we shall just construct the path from w_1 to x .

Both w_1 and x are in $P(v_1)$, and we give a general procedure for constructing a path between any two such vertices. The images under the inclusion ι of the triangles in $N(v_1)$ encircle $\iota(v_1)$. Suppose w_1 is contained in T_1 , and x is contained in T_2 . Traversing the triangles in a clockwise order from T_1 to T_2 gives one list of triangles, and traversing in a counterclockwise order gives another. Let $T_1, Q_1, \dots, Q_\ell, T_2$ be the shorter of these two lists, with a random choice made if the two lists are the same length. Choose a random vertex a_i in each Q_i , and let $a_0 = w_1$ and $a_{\ell+1} = x$. We thus have a vertex representing each triangle in the list. Our path will consist of a sequence of segments from each representative to the next.

Note that all of the triangles are distinct, except if $T_1 = T_2$ and the list is of length 2. We suppose for now that we have two vertices a_i and a_{i+1} in distinct triangles, and we deal with the degenerate case later. The two triangles in question

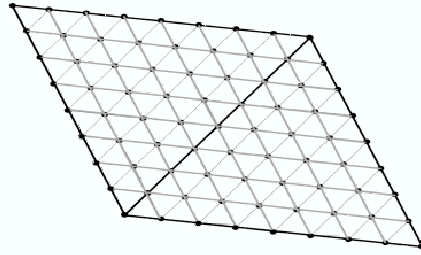


FIG. 6.2. An illustration of how the grid graph exists as a subgraph of the union of two adjacent subdivided triangles.

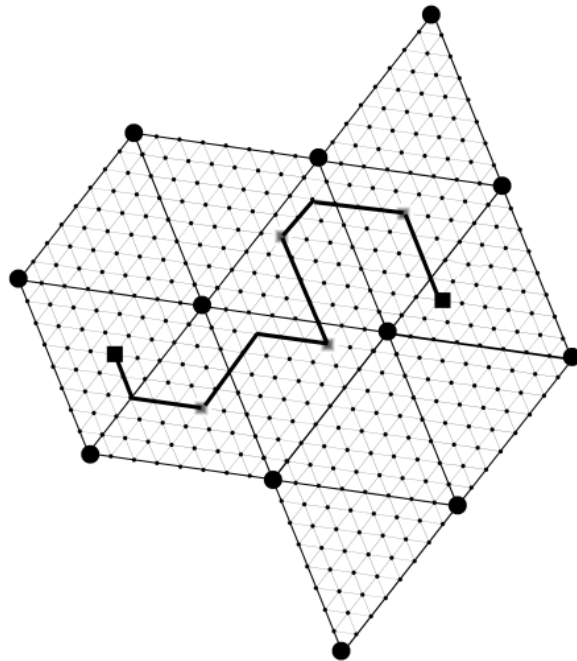


FIG. 6.3. The entire construction illustrated for a given edge of the original graph.

are adjacent, and their union contains a grid graph as a subgraph. (See Figure 6.2.) Given two vertices in a grid, there is a unique path between them that one obtains by first moving horizontally and then vertically, and another that one obtains by moving vertically and then horizontally. (These two coincide if there is a line connecting the two points.) Randomly choose one of these two paths. This is the path connecting a_i to a_{i+1} . If a_i and a_{i+1} lie in the same triangle, randomly choose one of the two adjacent triangles to form a grid, and then use the above construction. Composing the paths between each a_i and a_{i+1} completes the construction of π_E . The entire construction is illustrated in Figure 6.3.

We now consider the Rayleigh quotients for the three graphs that we have con-

structed. After k hexagonal refinements, every edge in G is split into $r = 2^k$ pieces, every triangle gets replaced with r^2 smaller triangles, and the number of vertices grows quadratically in r . A vector $y \in \mathbb{R}^{|V_{G'}|}$ that assigns a value to each vertex in G' gives the Rayleigh quotient

$$R(G') = \frac{\sum_{(i,j) \in E_{G'}} (y_i - y_j)^2}{y^T y}.$$

This induces a vector on the vertices of H by restriction. The probability, taken over the random choices in the construction of π_V and π_E , that a given edge of G' appears on the path representing a given edge e of G is zero if it is not in $P(\alpha)$ with α equal to one of the endpoints of e , and at most $O(1/r)$ otherwise. Since the maximum degree of a vertex in G is assumed constant, the expected number of times that a given edge of G' occurs in H is $O(1/r)$. Every vertex in G' is selected as a representative of a vertex in G with probability $\Theta(1/r^2)$. It thus follows that

$$(6.2) \quad \mathbb{E} \left[\sum_{(i,j) \in E_H} (y_i - y_j)^2 \right] \leq O\left(\frac{1}{r}\right) \sum_{(i,j) \in E_{G'}} (y_i - y_j)^2$$

and

$$(6.3) \quad \mathbb{E} \left[\sum_{i \in V_G} w_i y_{\pi_V(i)}^2 \right] = \Theta\left(\frac{1}{r^2}\right) \sum_{i \in V_{G'}} y_i^2,$$

where the expectations are taken over the random choices in the construction of (π_V, π_E) , and the w_i are any weights that are bounded above and below by positive constants.

Let \bar{y} be the vector in $\mathbb{R}^{|V_G|}$ whose i th coordinate is $y_{\pi_V(i)}$. Each coordinate \bar{y}_i of \bar{y} is chosen independently from a distinct set S_i of the coordinates of y , and every coordinate is contained in one of these sets. Let $s_i = |S_i|$, let $s_{\min} = \min_i s_i$, and take W to be the diagonal matrix whose i th diagonal entry w_i equals s_i/s_{\min} . The probability that a given vertex in S_i is selected equals $1/s_i$, so we have that

$$\mathbb{E} \left[\sum_{j \in V_G} w_j \bar{y}_j \right] = \sum_{k \in V_{G'}} y_k = 0.$$

(The need to weight the terms on the left-hand side of this expression by the w_i is what will necessitate the use of the weighted Laplacian in our proof.) The size of each S_i is approximately proportional to the degree of the i th vertex of G , so the w_i are all bounded above by a constant, and they are all at least one by definition. The eigenvalue $\tilde{\lambda}_2^W(G)$ of the weighted Laplacian is thus within a constant factor of the standard Fiedler value $\lambda_2(G)$.

Let z be the vector

$$z = \bar{y} - \left(\frac{\sum_i w_i \bar{y}_i}{\sum_i w_i} \right) \mathbf{1},$$

so that z differs from \bar{y} by a multiple of the all-ones vector and is orthogonal to $W\mathbf{1}$. By applying Lemma 6.3 to (6.3), we obtain

$$(6.4) \quad \mathbb{E} \left[\sum_{i \in V_G} w_i z_i^2 \right] \geq \left(1 - \frac{w_{\max}}{\sum_i w_i} \right) \mathbb{E} \left[\sum_{i \in V_G} w_i \bar{y}_i^2 \right] = \Theta\left(\frac{1}{r^2}\right) \sum_{i \in V_{G'}} y_i^2.$$

Multiplying the inequalities in (6.2) and (6.4) by the appropriate factors and combining them yields

$$(6.5) \quad O(r) \left(\sum_{(i,j) \in E_{G'}} (y_i - y_j)^2 \right) \cdot \mathbb{E} \left[\sum_{i \in V_G} w_i z_i^2 \right] \geq \left(\sum_{i \in V_{G'}} y_i^2 \right) \cdot \mathbb{E} \left[\sum_{(i,j) \in E_H} (y_i - y_j)^2 \right].$$

This implies that there exists some choice of (π_V, π_E) for which the left-hand side of (6.5) is greater than or equal to the right-hand side, in which case we would have

$$(6.6) \quad \frac{\sum_{(i,j) \in E_H} (y_i - y_j)^2}{\sum_{i \in V_G} w_i z_i^2} \leq O(r) \frac{\sum_{(i,j) \in E_{G'}} (y_i - y_j)^2}{\sum_{i \in V_{G'}} y_i^2} = O(r)R(G').$$

Now suppose that we assign to each vertex $v \in V_G$ the value assumed by y at $\pi_V(v)$. Using the fact that the maximum degree of a vertex is bounded, so that there are $O(1)$ triangles surrounding any vertex in G , we see that every path representing an edge is of length $O(r)$. We note that if i_1, \dots, i_s is a sequence of vertices,

$$(y_{i_s} - y_{i_1})^2 \leq s \sum_{a=1}^{s-1} (y_{i_{a+1}} - y_{i_a})^2.$$

As such, we have

$$(6.7) \quad \sum_{(i,j) \in E_G} (y_{\pi_V(i)} - y_{\pi_V(j)})^2 \leq O(r) \sum_{(i,j) \in E_H} (y_i - y_j)^2.$$

Since z is obtained from \bar{y} by subtracting a multiple of the all-ones vector,

$$z_i - z_j = y_{\pi_V(i)} - y_{\pi_V(j)}$$

for any i and j . Plugging this into (6.7) gives

$$\sum_{(i,j) \in E_G} (z_i - z_j)^2 \leq O(r) \sum_{(i,j) \in E_H} (y_i - y_j)^2,$$

and applying this to the inequality in (6.6) yields

$$\frac{\sum_{(i,j) \in E_G} (z_i - z_j)^2}{\sum_{i \in V_G} w_i z_i^2} \leq O(r^2)R(G').$$

We have thus constructed an assignment of values to the vertices of G that is orthogonal to the vector $W\mathbf{1}$ and produces a weighted Rayleigh quotient of $O(r^2)R(G')$. If we choose the y_i to be the values that give the Fiedler value of G' , we thus obtain, by (6.1) and the fact that the w_i are $\Theta(1)$,

$$\lambda_2(G) = \Theta(1)\tilde{\lambda}_2^W(G) \leq O(r^2)\lambda_2(G').$$

Since the number of vertices in G' grows as r^2 times the number of vertices in G , this completes the proof of Lemma 5.2. \square

Acknowledgments. I would like to thank Dan Spielman and Shang-Hua Teng for introducing me to this problem and for several extremely helpful conversations. I would also like to thank Nathan Dunfield, Phil Bowers, and Ken Stephenson for their guidance on the theory of circle packings, and Christopher Mihelich for his help in simplifying the proof of Lemma 6.3. I would also like to thank the anonymous referees for their extremely helpful comments and suggestions.

REFERENCES

- [1] E. M. ANDREEV, *Convex polyhedra in Lobachevskii space*, Math. USSR Sbornik, 10 (1970), pp. 413–440.
- [2] N. ALON, P. SEYMOUR, AND R. THOMAS, *A separator theorem for nonplanar graphs*, J. Amer. Math. Soc., 3 (1990), pp. 801–808.
- [3] C. J. ALPERT AND A. B. KAHNG, *Recent directions in netlist partitioning: A survey*, Integration: The VLSI Journal, 19 (1995), pp. 1–81.
- [4] A. F. BEARDON AND K. STEPHENSON, *The uniformization theorem for circle packings*, Indiana Univ. Math. J., 39 (1990), pp. 1383–1425.
- [5] P. BOWERS AND K. STEPHENSON, *Uniformizing dessins and Belyi maps via circle packing*, Mem. Amer. Math. Soc., 170 (2004).
- [6] P. CHAN, M. SCHLAG, AND J. ZIEN, *Spectral k -way ratio-cut partitioning and clustering*, IEEE Trans. Computer-Aided Design Integrated Circuits Systems, 13 (1994), pp. 1088–1096.
- [7] T. CHAN AND D. C. RESASCO, *A framework for the analysis and construction of domain decomposition preconditioners*, in Proceedings of the First International Symposium on Domain Decomposition Methods, Paris, 1987, R. Glowinski et al., eds., SIAM, Philadelphia, 1988, pp. 217–230.
- [8] T. CHAN AND B. SMITH, *Domain decomposition and multigrid algorithms for elliptic problems on unstructured meshes*, Contemp. Math., (1993), pp. 1–14.
- [9] F. R. K. CHUNG, *Spectral Graph Theory*, AMS, Providence, RI, 1997.
- [10] H. DJIDJEV, *A linear algorithm for partitioning graphs*, C. R. Acad. Bulgare Sci., 35 (1982), pp. 1053–1056.
- [11] H. DJIDJEV AND J. REIF, *An efficient algorithm for the genus problem with explicit construction of forbidden subgraphs*, in Proceedings of the 23rd Annual ACM Symposium on the Theory of Computing, New Orleans, LA, 1991, ACM, New York, pp. 337–348.
- [12] I. FILOTTI, G. MILLER, AND J. REIF, *On determining the genus of a graph in $o(v^{O(g)})$ steps*, in Proceedings of the 11th Annual ACM Symposium on the Theory of Computing, Atlanta, GA, 1979, ACM, New York, pp. 27–37.
- [13] O. FORSTER, *Lectures on Riemann Surfaces*, Springer-Verlag, New York, 1981.
- [14] J. GILBERT, J. HUTCHINSON, AND R. TARJAN, *A separation theorem for graphs of bounded genus*, J. Algorithms, 5 (1984), pp. 391–407.
- [15] S. GUATTERY AND G. L. MILLER, *On the performance of spectral graph partitioning methods*, in Proceedings of the Sixth Annual ACM-SIAM Symposium on Discrete Algorithms, San Francisco, CA, 1995, SIAM, Philadelphia, 1995, pp. 233–242.
- [16] L. HAGEN AND A. B. KAHNG, *New spectral methods for ratio cut partitioning and clustering*, IEEE Trans. Computer-Aided Design, 11 (1992), pp. 1074–1085.
- [17] Z.-X. HE AND O. SCHRAMM, *Fixed points, Koebe uniformization, and circle packings*, Ann. Math., 137 (1993), pp. 369–406.
- [18] P. KOEBE, *Kontaktprobleme der Konformen Abbildung*, Ber. Sächs. Akad. Wiss. Leipzig, Math.-Phys. Kl., 88 (1936), pp. 141–164.
- [19] R. J. LIPTON AND R. E. TARJAN, *A separator theorem for planar graphs*, SIAM J. Appl. Math., 36 (1979), pp. 177–189.
- [20] G. MCCAUGHAN, *A recurrence/transience result for circle packings*, Proc. Amer. Math. Soc., 126 (1998), pp. 3647–3656.
- [21] B. MOHAR, *A linear time algorithm for embedding graphs in an arbitrary surface*, SIAM J. Discrete Math., 12 (1999), pp. 6–26.
- [22] G. MIHAIL, *Conductance and convergence of Markov chains—A combinatorial treatment of expanders*, in Proceedings of the 29th Annual IEEE Conference on Foundations of Computer Science, 1989, IEEE Press, Piscataway, NJ, pp. 526–531.
- [23] B. RODIN AND D. SULLIVAN, *The convergence of circle packings to the Riemann mapping*, J. Differential Geometry, 26 (1987), pp. 349–360.
- [24] H. SIMON, *Partitioning of unstructured problems for parallel processing*, Comput. Systems Engineering, 2 (1991), pp. 135–148.

- [25] D. A. SPIELMAN AND S.-H. TENG, *Spectral partitioning works: Planar graphs and finite element meshes*, in Proceedings of the 37th Annual IEEE Conference on Foundations of Computer Science, Burlington, VT, 1996, IEEE Press, Piscataway, NJ, 1996, pp. 96–105.
- [26] K. STEPHENSON, *Circle packing and discrete analytic function theory*, in Handbook of Complex Analysis, Vol. 1: Geometric Function Theory, R. Kühnau, ed., Elsevier, New York, Chapter 11.
- [27] C. THOMASSEN, *The graph genus problem is NP-complete*, J. Algorithms, 10 (1989), pp. 568–576.
- [28] W. THURSTON, *The Geometry and Topology of 3-Manifolds*, Princeton University Notes, preprint.
- [29] R. D. WILLIAMS, *Performance of dynamic load balancing algorithms for unstructured mesh calculations*, Concurrency: Pract. Exp., 3 (1991), pp. 457–481.

USING NONDETERMINISM TO AMPLIFY HARDNESS*

ALEXANDER HEALY[†], SALIL VADHAN[†], AND EMANUELE VIOLA[†]

Abstract. We revisit the problem of hardness amplification in \mathcal{NP} , as recently studied by O’Donnell [*J. Comput. System Sci.*, 69 (2004), pp. 68–94]. We prove that if \mathcal{NP} has a balanced function f such that any circuit of size $s(n)$ fails to compute f on a $1/\text{poly}(n)$ fraction of inputs, then \mathcal{NP} has a function f' such that any circuit of size $s'(n) = s(\sqrt{n})^{\Omega(1)}$ fails to compute f' on a $1/2 - 1/s'(n)$ fraction of inputs. In particular,

1. if $s(n) = n^{\omega(1)}$, we amplify to hardness $1/2 - 1/n^{\omega(1)}$;
2. if $s(n) = 2^{n^{\Omega(1)}}$, we amplify to hardness $1/2 - 1/2^{n^{\Omega(1)}}$;
3. if $s(n) = 2^{\Omega(n)}$, we amplify to hardness $1/2 - 1/2^{\Omega(\sqrt{n})}$.

Our results improve those of O’Donnell, which amplify to $1/2 - 1/\sqrt{n}$. O’Donnell also proved that no construction of a certain general form could amplify beyond $1/2 - 1/n$. We bypass this barrier by using both *derandomization* and *nondeterminism* in the construction of f' .

We also prove impossibility results demonstrating that both our use of nondeterminism and the hypothesis that f is balanced are necessary for “black-box” hardness amplification procedures (such as ours).

Key words. average-case complexity, hardness amplification, pseudorandom generators for space-bounded computation, noise stability

AMS subject classifications. 68Q01, 68Q17

DOI. 10.1137/S0097539705447281

1. Introduction. *Average-case complexity* is a fundamental topic in complexity theory, whose study has at least two distinct motivations. On the one hand, it may provide more meaningful explanations than worst-case complexity about the intractability of problem instances actually encountered in practice. On the other hand, it provides us with methods for generating hard instances, allowing us to harness intractability for useful ends such as cryptography and derandomization.

One of the goals of this area is to establish connections between average-case complexity and worst-case complexity. While this has been accomplished for high complexity classes such as $\#\mathcal{P}$ and $\mathcal{EXPTIME}$ (e.g., [2, 4, 7, 8, 25, 33, 35, 37]), it remains a major open question for \mathcal{NP} . In fact, there are results showing that such connections for \mathcal{NP} are unlikely to be provable using the same kinds of techniques used for the high complexity classes [6, 9, 37, 38].

A more modest goal is “hardness amplification,” where we seek to establish connections between “mild” average-case complexity and “strong” average-case complexity. That is, given a problem for which a nonnegligible fraction of inputs are “hard,” can we obtain a problem for which almost all inputs are hard? To make this precise, let us define “hard.”

*Received by the editors October 8, 2004; accepted for publication (in revised form) July 30, 2005; published electronically February 21, 2006. An extended abstract of this paper appeared in *Proceedings of the 36th Annual ACM Symposium on the Theory of Computing*; see [17].

<http://www.siam.org/journals/sicomp/35-4/44728.html>

[†]Division of Engineering and Applied Sciences, Harvard University, Cambridge, MA 02138 (ahealy@fas.harvard.edu, salil@eecs.harvard.edu, viola@eecs.harvard.edu). The research of the first author was supported in part by NSF grant CCR-0205423 and by a Sandia Fellowship. The research of the second author was supported by NSF grant CCR-0133096, ONR grant N00014-04-1-0478, a Sloan Research Fellowship, and by US-Israel BSF grant 2002246. This author’s work was done in part while a fellow at the Radcliffe Institute for Advanced Study at Harvard University. The research of the third author was supported by NSF grant CCR-0133096 and US-Israel BSF grant 2002246.

DEFINITION 1.1. For $\delta \in [0, 1/2]$, a function $f : \{0, 1\}^n \rightarrow \{0, 1\}$ is δ -hard for size s if every circuit of size s fails to compute f on at least a δ fraction of inputs.

Note that the maximum value of the hardness parameter δ is $1/2$ because f is Boolean (and so can trivially be computed with error probability at most $1/2$). This notion of hardness is fairly standard (e.g., in the literature on derandomization starting from [30]), but we remark that it differs from Levin’s notion of average-case complexity [24] in several ways. Most importantly, Levin’s formulation corresponds to algorithms that always either give the correct answer or say “don’t know,” whereas we consider even “heuristic” algorithms that can make arbitrary errors. (See Impagliazzo’s survey [19].)

The *hardness amplification problem* is to convert a function f that is δ -hard for size s into a function f' that is $(1/2 - \epsilon)$ -hard for size polynomially related to s . Commonly, $\delta = 1/\text{poly}(n)$, and the aim is to make $\epsilon = \epsilon(n)$ vanish as quickly as possible.

The standard approach to hardness amplification employs Yao’s XOR lemma [39] (see [15]): Given a mildly hard-on-average function $f : \{0, 1\}^n \rightarrow \{0, 1\}$, we define $f' : \{0, 1\}^{n \cdot k} \rightarrow \{0, 1\}$ by

$$f'(x_1, \dots, x_k) \stackrel{\text{def}}{=} f(x_1) \oplus f(x_2) \oplus \dots \oplus f(x_k).$$

The XOR Lemma says that the hardness of f' approaches $1/2$ exponentially fast with k . More precisely, this can be stated as follows.

YAO’S XOR LEMMA. If f is δ -hard for size $s(n) \geq n^{\omega(1)}$ and $k \leq \text{poly}(n)$, then f' is $(1/2 - 1/2^{\Omega(\delta k)} - 1/s')$ -hard for size $s'(n \cdot k) = s(n)^{\Omega(1)}$.

In particular, taking $k = \Theta(n/\delta)$, the amplified hardness is dominated by the $1/s'$ term. That is, we can amplify to hardness $(1/2 - \epsilon)$, where ϵ is polynomially related to the (reciprocal of the) circuit size for which f was hard. (Note, however, that we should measure $\epsilon = \epsilon(n')$ as a function of the new input length $n' = n \cdot k$, so when $k = n$, the hardness is actually $1/2 - 1/s(\sqrt{n'})^{\Omega(1)}$.)

However, if we are interested in hardness amplification within \mathcal{NP} (i.e., f and f' are characteristic functions of languages in \mathcal{NP}), we cannot use the XOR lemma; it does not ensure that f' is in \mathcal{NP} when f is in \mathcal{NP} . Hardness amplification within \mathcal{NP} was first addressed in a recent paper of O’Donnell [31], which is the starting point for our work.

1.1. O’Donnell’s hardness amplification. To ensure that the new function f' is in \mathcal{NP} when f is in \mathcal{NP} , O’Donnell [31] was led to study constructions of the form

$$(1) \quad f'(x_1, \dots, x_k) \stackrel{\text{def}}{=} C(f(x_1), f(x_2), \dots, f(x_k)),$$

where C is an efficiently computable *monotone* function. The monotonicity of C ensures that f' is in \mathcal{NP} when f is in \mathcal{NP} . However, we are left with the task of choosing such a function C and proving that it indeed amplifies hardness.

Remarkably, O’Donnell was able to precisely characterize the amplification properties of construction (1) in terms of a combinatorial property of the combining function C , called its *expected bias*. (The actual definition is not needed for this discussion, but can be found in section 3.) By finding a monotone combining function in which this expected bias is small, he obtained the first positive result on hardness amplification in \mathcal{NP} , given next.

O'DONNELL'S THEOREM (see [31]). *If \mathcal{NP} contains a balanced function that is $1/\text{poly}(n)$ -hard for polynomial-size circuits, then \mathcal{NP} contains a function that is $(1/2 - 1/n^{1/2-\alpha})$ -hard for polynomial-size circuits (where α is an arbitrarily small positive constant).*

However, the amplification provided by O'Donnell's theorem is not as strong as what the XOR lemma gives. It is limited to $1/2 - 1/\sqrt{n}$, regardless of the circuit size s for which the original function is hard, even if s is exponentially large. The XOR lemma, on the other hand, amplifies to $1/2 - 1/s^{\Omega(1)}$. O'Donnell showed that this difference is inherent—no construction of the form (1) with a monotone combining function C can always amplify hardness to better than $1/2 - 1/n$.¹

1.2. Our result. In this paper, we manage to amplify hardness within \mathcal{NP} beyond the $1/2 - 1/n$ barrier, as described in the following.

THEOREM 1.2 (main theorem). *If \mathcal{NP} contains a balanced function that is $1/\text{poly}(n)$ -hard for circuits of size $s(n)$, then \mathcal{NP} contains a function that is $(1/2 - 1/s'(n))$ -hard for circuits of size $s'(n)$ for some function² $s'(n) = s(\sqrt{n})^{\Omega(1)}$. In particular,*

1. *if $s(n) = n^{\omega(1)}$, we amplify to hardness $1/2 - 1/n^{\omega(1)}$;*
2. *if $s(n) = 2^{n^{\Omega(1)}}$, we amplify to hardness $1/2 - 1/2^{n^{\Omega(1)}}$;*
3. *if $s(n) = 2^{\Omega(n)}$, we amplify to hardness $1/2 - 1/2^{\Omega(\sqrt{n})}$.*

Items 1–3 match the parameters of the Yao's XOR lemma. However, subsequent “derandomizations” of the XOR lemma [18, 20] actually amplify up to $1/2 - 1/2^{\Omega(n)}$ rather than just $1/2 - 1/2^{\Omega(\sqrt{n})}$ in the case $s(n) = 2^{\Omega(n)}$. This gap is *not* inherent in our approach and, as mentioned below, would be eliminated given a corresponding improvement in one of the tools we employ.

Of course, our construction cannot be of the form in construction (1). Below we describe our two main points of departure.

1.3. Techniques. To explain how we bypass it, we first look more closely at the source of the $1/2 - 1/n$ barrier. The actual barrier is $1/2 - 1/k$, where k is the input length of the monotone combining function C . (This is based on a result of [22]; see [31].) Since in construction (1), f' has input length $n' = n \cdot k \geq k$, it follows that we cannot amplify beyond $1/2 - 1/n'$.

Derandomization. Given the above, our first idea is to break the link between the input length of f' and the input length of the combining function C . We do this by *derandomizing* O'Donnell's construction. That is, the inputs x_1, \dots, x_k are no longer taken independently (as in construction (1)), but are generated pseudorandomly from a short seed of length $n' \ll k$, which becomes the actual input to f' . Our method for generating the x_i 's is based on combinatorial designs (as in the Nisan–Wigderson generator [30]) and Nisan's pseudorandom generator for space-bounded computation [29]; it reduces the input length of f' from $n \cdot k$ to $n' = O(n^2 + \log^2 k)$. We stress that this derandomization is unconditional, i.e., requires no additional complexity assumption. We also remark that it is the quadratic seed length of Nisan's generator that limits our amplification to $1/2 - 1/2^{\Omega(\sqrt{n})}$ rather than $1/2 - 1/2^{\Omega(n)}$ in part 3 of our main

¹The gap between O'Donnell's positive result of $1/2 - 1/\sqrt{n}$ and his negative result of $1/2 - 1/n$ is not significant for what follows, and in particular it will be subsumed by our improvements.

²In the rest of the paper, we more compactly write “for circuits of size $s'(n) = s(\sqrt{n})^{\Omega(1)}$,” where the $\Omega(1)$ is to be interpreted as a fixed constant (possibly depending on previously quantified constants).

theorem, and thus any improvement in Nisan’s generator would yield a corresponding improvement in our result.

Similar derandomizations have previously been achieved for Yao’s XOR lemma by Impagliazzo [18] and Impagliazzo and Wigderson [20]. The analysis of such derandomizations is typically tailored to a particular proof, and indeed [18, 20] each gave a new proof of the XOR lemma for that purpose. In our case, we do not know how to derandomize O’Donnell’s original proof, but instead manage to derandomize a different proof due to Trevisan [34].

Our derandomization allows for k to be larger than the input length of f' , and hence we can go beyond the $1/2 - 1/n'$ barrier. Indeed, by taking k to be a sufficiently large polynomial, we amplify to $1/2 - 1/(n')^c$ for any constant c .

Using nondeterminism. To amplify further, it is tempting to take k superpolynomial in the input length of f' . But then we run into a different problem: how do we ensure that f' is in \mathcal{NP} ? The natural algorithm for f' requires running the algorithm for f on k inputs.

To overcome this difficulty, we observe that we need only give an efficient *nondeterministic* algorithm for f' . Each nondeterministic path may involve only polynomially many evaluations of f , while the global outcome $f'(x)$ depends on exponentially many evaluations. To implement this idea, we exploit the specific structure of the combining function C . Namely, we (like O’Donnell) use the Tribes function of Ben-Or and Linial [5], which is a monotone DNF with clauses of size $O(\log k)$. Thus, the nondeterministic algorithm for f' can simply guess a satisfied clause and (nondeterministically) evaluate f on the $O(\log k)$ corresponding inputs.

1.4. Other results. We also present some complementary negative results:

- We show that the assumption that the original hard function is balanced is necessary, in the sense that no monotone “black-box” hardness amplification can amplify unbalanced functions of unknown bias (or even improve their bias).³
- We show that our use of nondeterminism is necessary, in the sense that any “black-box” hardness amplification in which each evaluation of f' is a monotone function of at most k evaluations of f can amplify hardness to at most $1/2 - 1/k$.

Informally, a “black-box” hardness amplification is one in which the construction of the amplified function f' from f utilizes f only as an oracle and is well defined for any function f (regardless of whether or not it is in \mathcal{NP}). Moreover, the correctness of the construction is proved by a generic reduction that converts any *oracle* A (regardless of its circuit size) that computes f' well on average (e.g., with probability $1/2 + \epsilon$ over random choice of input) into one that computes f much better on average (e.g., with probability $1 - o(1)$ over random input). (A formal definition is given in section 7.1.) We note that most results on hardness amplification against circuits, including ours, are black-box (though there have been some recent results using non-black-box techniques in hardness amplification against *uniform* algorithms; see [21, 35]).

Our framework also gives a new proof of the hardness amplification by Impagliazzo and Wigderson [20]. Our proof is simpler, and in particular its analysis does not employ the Goldreich–Levin [14] step.

³We note that there do exist balanced \mathcal{NP} -complete problems, as observed by Barak and reported in [10], but this has no direct implication for us because we are studying the average-case complexity of \mathcal{NP} .

1.5. Organization. The rest of the paper is organized as follows. In section 2, we discuss some preliminaries. In section 3, we review existing results on hardness amplification in \mathcal{NP} . In section 4, we present our main results and new techniques. In section 5 we treat the details of the proof of our main theorem. In section 6 we show how we could amplify to $1/2 - 1/2^{\Omega(n)}$, given an improvement in the pseudorandom generator we use, and we also give a new proof of the hardness amplification by Impagliazzo and Wigderson [20]. In section 7 we discuss some limitations of monotone hardness amplification; in particular we show a sense in which the hypothesis that the starting function is balanced is necessary, and also that the use of nondeterminism is necessary.

2. Preliminaries. We denote the uniform distribution on $\{0, 1\}^n$ by U_n . If U_n occurs more than once in the same expression, it is understood that these all represent the same random variable; for example, $U_n \cdot f(U_n)$ denotes the random variable obtained by choosing X uniformly at random in $\{0, 1\}^n$ and outputting $X \cdot f(X)$ (where \cdot means concatenation).

DEFINITION 2.1. *Let X and Y be two random variables taking values over the same set S . Then the statistical difference between X and Y is*

$$\Delta(X, Y) \stackrel{\text{def}}{=} \max_{T \subseteq S} \left| \Pr[X \in T] - \Pr[Y \in T] \right|.$$

We view probabilistic functions as functions of two inputs, e.g., $h(x; r)$, the first being the input to the function and the second being the randomness. (Deterministic functions may be thought of as probabilistic functions that ignore the randomness.) For notational convenience, we will often omit the second input to a probabilistic function, e.g., writing $h(x)$ instead of $h(x; r)$, in which case we view $h(x)$ as the random variable $h(x; U_{|r|})$.

DEFINITION 2.2. *The bias of a 0-1 random variable X is*

$$\text{Bias}[X] \stackrel{\text{def}}{=} \left| \Pr[X = 0] - \Pr[X = 1] \right| = 2 \cdot \Delta(X, U_1).$$

Analogously, the bias of a probabilistic function $f : \{0, 1\}^n \rightarrow \{0, 1\}$ is

$$\text{Bias}[f] \stackrel{\text{def}}{=} \left| \Pr[f(U_n) = 0] - \Pr[f(U_n) = 1] \right|,$$

where the probabilities are taken over both the input chosen according to U_n and the coin tosses of f . We say that f is balanced when $\text{Bias}[f] = 0$.

Note that the bias of a random variable is a quantity between 0 and 1.

We say that the random variables X and Y are ϵ -indistinguishable for size s if for every circuit C of size s ,

$$\left| \Pr_X[C(X) = 1] - \Pr_Y[C(Y) = 1] \right| \leq \epsilon.$$

We will routinely use the following connection between hardness and indistinguishability.

LEMMA 2.3 (see [39]). *Let $h : \{0, 1\}^n \rightarrow \{0, 1\}$ be any probabilistic function. If the distributions $U_n \cdot h(U_n)$ and $U_n \cdot U_1$ are ϵ -indistinguishable for size s , then h is $(1/2 - \epsilon)$ -hard for size $s - O(1)$. Conversely, if h is $(1/2 - \epsilon)$ -hard for size s , then the distributions $U_n \cdot h(U_n)$ and $U_n \cdot U_1$ are ϵ -indistinguishable for size $s - O(1)$.*

Finally, whenever we amplify the hardness of a function $f : \{0, 1\}^n \rightarrow \{0, 1\}$ that is hard for circuits of size $s(n)$, we assume that $s(n)$ is well behaved in the sense that

it is computable in time $\text{poly}(n)$ and $s(cn) = s(n)^{O(1)}$, for all constants $c > 0$. Most natural functions smaller than 2^n , such as $n^k, 2^{\log^k n}, 2^{n^\epsilon}, 2^{\epsilon n}$, are well behaved in this sense.

3. Overview of previous hardness amplification in \mathcal{NP} . In this section we review the essential components of existing results on hardness amplification in \mathcal{NP} . We then discuss the limitations of these techniques. By the end of this section, we will have sketched the main result of O’Donnell [31], following the approach of Trevisan [34]. We outline this result in a way that will facilitate the presentation of our results in subsequent sections.

Let $f : \{0, 1\}^n \rightarrow \{0, 1\}$ be an average-case hard function, and let $C : \{0, 1\}^k \rightarrow \{0, 1\}$ be any function. In [31], O’Donnell studies the hardness of functions of the form

$$C \circ f^{\otimes k} : (\{0, 1\}^n)^k \rightarrow \{0, 1\},$$

where $f^{\otimes k}(x_1, \dots, x_k) \stackrel{\text{def}}{=} (f(x_1), \dots, f(x_k))$ and \circ denotes composition. That is,

$$(C \circ f^{\otimes k})(x_1, \dots, x_k) \stackrel{\text{def}}{=} C(f(x_1), \dots, f(x_k)).$$

In order to ensure that $C \circ f^{\otimes k} \in \mathcal{NP}$ whenever $f \in \mathcal{NP}$, O’Donnell chooses C to be a polynomial-time computable *monotone* function. (Indeed, it is not hard to see that a monotone combination of \mathcal{NP} functions is itself in \mathcal{NP} .)

O’Donnell characterizes the hardness of $C \circ f^{\otimes k}$ in terms of a combinatorial property of the combining function C , called its *expected bias* (which we define later). We will now review the key steps in establishing this characterization and O’Donnell’s final amplification theorem.

Step 1: Impagliazzo’s hardcore sets. An important tool for establishing this connection is the hardcore set lemma of Impagliazzo [18], which allows us to pass from computational hardness to information-theoretic hardness.

DEFINITION 3.1. *We say that a (probabilistic) function $g : \{0, 1\}^n \rightarrow \{0, 1\}$ is δ -random if g is balanced and there exists a subset $H \subseteq \{0, 1\}^n$ with $|H| = 2\delta 2^n$ such that $g(x) = U_1$ (i.e., a coin flip) for $x \in H$ and $g(x)$ is deterministic for $x \notin H$.*

Thus, a δ -random function has a set of relative size 2δ on which it is information-theoretically unpredictable. Note that in the above definition we require g to be balanced. This will be convenient when dealing with functions f that are balanced.

The following version of the Impagliazzo hardcore set lemma says that any balanced δ -hard function $f : \{0, 1\}^n \rightarrow \{0, 1\}$ has a hardcore set $H \subseteq \{0, 1\}^n$ of density $\approx 2\delta$ such that f is very hard on average on H . Thus, f looks like a δ -random function to small circuits (cf. Lemma 2.3). (Following subsequent works, our formulation of Impagliazzo’s lemma differs from the original in several respects.)

LEMMA 3.2 (see [18, 23, 33, 31]). *For any function $f : \{0, 1\}^n \rightarrow \{0, 1\}$ that is balanced and δ -hard for size s , there exists a δ' -random function $g : \{0, 1\}^n \rightarrow \{0, 1\}$ such that $X \cdot f(X)$ and $X \cdot g(X)$ are ϵ -indistinguishable for size $\Omega(s\epsilon^2 / \log(1/\delta))$, with $\delta/2 \leq \delta' \leq \delta$, where $X \equiv U_n$.*

In particular, by a standard hybrid argument (see, e.g., [13]),

$$X_1 \cdots X_k \cdot f(X_1) \cdots f(X_k) \quad \text{and} \quad X_1 \cdots X_k \cdot g(X_1) \cdots g(X_k)$$

are $k\epsilon$ -indistinguishable for size $\Omega(s\epsilon^2 / \log(1/\delta))$, where the X_i ’s are uniform and independent.

Step 2: expected bias. By the above, proving the computational hardness of $C \circ f^{\otimes k}$ reduces to calculating the information-theoretic hardness of $C \circ g^{\otimes k}$ for some δ' -random g . It turns out that information-theoretic hardness can be characterized by the following quantity.

DEFINITION 3.3. *Let $h : \{0, 1\}^n \rightarrow \{0, 1\}$ be any probabilistic function. We define the expected bias of h by*

$$\text{ExpBias}[h] \stackrel{\text{def}}{=} \mathbb{E}_{x \leftarrow U_n} [\text{Bias}[h(x)]],$$

where $\text{Bias}[h(x)]$ is taken over the coin tosses of h .

It turns out that for any function $C : \{0, 1\}^k \rightarrow \{0, 1\}$ and any δ -random g , the quantity $\text{ExpBias}[C \circ g^{\otimes k}]$ does not depend on the particular choice of the δ -random function g ; indeed, it turns out to equal the quantity that O'Donnell [31] calls the ‘‘expected bias of C with respect to noise 2δ ’’ and denotes by $\text{ExpBias}_{2\delta}(C)$ in [31]. However, the more general notation we use will be useful in presenting our improvements.

The next lemma shows that information-theoretic hardness is equivalent to expected bias.

LEMMA 3.4. *For any probabilistic function $h : \{0, 1\}^n \rightarrow \{0, 1\}$,*

$$\Delta(U_n \cdot h(U_n), U_n \cdot U_1) = \frac{1}{2} \text{ExpBias}[h].$$

Proof. $\Delta(U_n \cdot h(U_n), U_n \cdot U_1) = \mathbb{E}_{x \leftarrow U_n} [\Delta(h(x), U_1)] = \mathbb{E}_{x \leftarrow U_n} [\text{Bias}[h(x)]/2] = \text{ExpBias}[h]/2.$ \square

In particular, for any circuit C (regardless of its size) we have $|\Pr[C(U_n \cdot h(U_n)) = 1] - \Pr[C(U_n \cdot U_1) = 1]| \leq \text{ExpBias}[h]/2$, and thus by Lemma 2.3, h is $(1/2 - \text{ExpBias}[h]/2)$ -hard for circuits of any size.

Now we characterize the hardness of $C \circ f^{\otimes k}$ in terms of expected bias. Specifically, by taking, say, $\epsilon = 1/s^{1/3}$ in Lemma 3.2 and using Lemmas 2.3 and 3.4, one can prove the following (we defer the details until the proof of the more general Lemma 5.2).

LEMMA 3.5 (see [31]). *Let $f : \{0, 1\}^n \rightarrow \{0, 1\}$ be balanced and δ -hard for size s , and let $C : \{0, 1\}^k \rightarrow \{0, 1\}$ be any function. Then there exists a δ' -random function $g : \{0, 1\}^n \rightarrow \{0, 1\}$, with $\delta/2 \leq \delta' \leq \delta$, such that $C \circ f^{\otimes k} : (\{0, 1\}^n)^k \rightarrow \{0, 1\}$ has hardness*

$$\frac{1}{2} - \frac{\text{ExpBias}[C \circ g^{\otimes k}]}{2} - \frac{k}{s^{1/3}}$$

for circuits of size $\Omega(s^{1/3}/\log(1/\delta)) - \text{size}(C)$, where $\text{size}(C)$ denotes the size of a smallest circuit computing C .

What makes this lemma so useful is that, as noted above, the quantity $\text{ExpBias}[C \circ g^{\otimes k}]$ is independent of the choice of the δ -random function g (using the fact that g is balanced, by definition of δ -random); hence the hardness of $C \circ f^{\otimes k}$ depends only on the combining function C and the hardness parameter δ . Thus, understanding the hardness of $C \circ f^{\otimes k}$ is reduced to analyzing a combinatorial property of the combining function C .

Step 3: noise stability. Unfortunately, it is often difficult to analyze the expected bias directly. However, the expected bias is closely related to the *noise stability*, a quantity that is more amenable to analysis and better studied (see, e.g., [31, 27]). The

noise stability of a function is (up to normalization) the probability that the value of the function is the same on two correlated inputs x and $x + \eta$, where x is a random input and η a random vector of noise.

DEFINITION 3.6. *The noise stability of C with respect to the noise δ , denoted $\text{NoiseStab}_\delta[C]$, is defined by*

$$\text{NoiseStab}_\delta[C] \stackrel{\text{def}}{=} 2 \cdot \Pr_{x,\eta}[C(x) = C(x \oplus \eta)] - 1,$$

where x is random, η is a vector whose bits are independently one with probability δ , and \oplus denotes bitwise XOR.

The following lemma from [31] bounds the expected bias of $C \circ g^{\otimes k}$ (and hence the hardness in Lemma 3.5) in terms of the noise stability of C .

LEMMA 3.7. *Let $g : \{0, 1\}^n \rightarrow \{0, 1\}$ be δ -random. Then*

$$\text{ExpBias}[C \circ g^{\otimes k}] \leq \sqrt{\text{NoiseStab}_\delta[C]}.$$

Combining this with Lemma 3.5, we find that the hardness of $C \circ f^{\otimes k}$ is at least (roughly) $1/2 - \sqrt{\text{NoiseStab}_\delta[C]}/2$. The next step is to exhibit a combining function C with a small noise stability (to ensure that the hardness of $C \circ f^{\otimes k}$ is as close to $1/2$ as possible). The following is shown in [31].

LEMMA 3.8 (see [31]). *For all $\delta > 0$, there exists a $k = \text{poly}(1/\delta)$ and a polynomial-time computable monotone function $C : \{0, 1\}^k \rightarrow \{0, 1\}$ with $\text{NoiseStab}_\delta[C] \leq 1/k^{\Omega(1)}$.*

Finally, by combining Lemmas 3.5, 3.7, and 3.8, we obtain the following weaker version of O’Donnell’s hardness amplification within \mathcal{NP} . (While in the introduction we mentioned a stronger version of O’Donnell’s result, which amplifies up to hardness $1/2 - 1/m^{1/2-\alpha}$ for every constant $\alpha > 0$, the following version will suffice as a starting point for our work. The loss in the amplification in this version comes from the fact that we did not specify the constants in Lemma 3.8.)

THEOREM 3.9 (see [31]). *If there is a balanced function $f : \{0, 1\}^n \rightarrow \{0, 1\}$ in \mathcal{NP} that is $1/\text{poly}(n)$ -hard for size $s(n)$, then there is a function $f' : \{0, 1\}^m \rightarrow \{0, 1\}$ in \mathcal{NP} that is $(1/2 - 1/m^{\Omega(1)})$ -hard for size $s(m^{\Omega(1)})^{\Omega(1)}$.*

Limitations of direct product constructions. O’Donnell also showed that Theorem 3.9 is essentially the best result that one can obtain using the techniques that we have described thus far. He showed that for all monotone combining functions C there is a δ -hard function f such that the hardness of $C \circ f^{\otimes k}$ is not much better than $1/2 - \text{NoiseStab}_\delta[C]/2$ (assuming that C is easily computable). This is problematic because the noise stability of monotone functions cannot become too small. Specifically, by combining a result from [22] with a Fourier characterization of noise stability, O’Donnell [31] proves the following theorem.

THEOREM 3.10 (see [22, 31]). *For every monotone function $C : \{0, 1\}^k \rightarrow \{0, 1\}$ and every $\delta > 0$,*

$$\text{NoiseStab}_\delta[C] \geq (1 - 2\delta) \cdot \Omega\left(\frac{\log^2 k}{k}\right).$$

Therefore, for any monotone $C : \{0, 1\}^k \rightarrow \{0, 1\}$ there is a δ -hard f such that $C \circ f^{\otimes k}$ does not have hardness $1/2 - \text{NoiseStab}_\delta[C]/2 \leq 1/2 - \Omega(1/k)$. Since $C \circ f^{\otimes k}$ takes inputs of length $m = n \cdot k \geq k$, this implies that we must employ a new technique to amplify beyond hardness $1/2 - \Omega(1/m)$.

4. Main theorem and overview. In this paper, we obtain the following improvement upon Theorem 3.9.

THEOREM 4.1 (main theorem, restated). *If there is a balanced function $f : \{0, 1\}^n \rightarrow \{0, 1\}$ in \mathcal{NP} that is $1/\text{poly}(n)$ -hard for size $s(n)$, then there is a function $f' : \{0, 1\}^m \rightarrow \{0, 1\}$ in \mathcal{NP} that is $(1/2 - 1/s(\sqrt{m})^{\Omega(1)})$ -hard for size $s(\sqrt{m})^{\Omega(1)}$.*⁴

We also show that the assumption that we start with a *balanced* function f is essential for a large class of hardness amplifications. Specifically, we show (section 7.1) that no monotone black-box hardness amplification can amplify the hardness of functions whose bias is unknown. Most hardness amplifications, including the one in this paper, are black-box.

We now elaborate on the two main techniques that allow us to prove Theorem 4.1. As explained in the introduction, these two techniques are derandomization and nondeterminism.

4.1. Derandomization. As in the previous section, let $f : \{0, 1\}^n \rightarrow \{0, 1\}$ be our hard function, and let $C : \{0, 1\}^k \rightarrow \{0, 1\}$ be a (monotone) combining function.

We will derandomize O’Donnell’s construction using an appropriately “pseudo-random” generator.

DEFINITION 4.2. *A generator is a function $G : \{0, 1\}^l \rightarrow (\{0, 1\}^n)^k$. We call l the seed length of G , and we often write $G(\sigma) = X_1 \cdots X_k$, with each $X_i \in \{0, 1\}^n$. G is explicitly computable if, given σ and $1 \leq i \leq k$, we can compute X_i in time $\text{poly}(l, \log k)$, where $G(\sigma) = X_1 \cdots X_k$.*

Instead of using the function $C \circ f^{\otimes k} : (\{0, 1\}^n)^k \rightarrow \{0, 1\}$, we take a generator $G : \{0, 1\}^l \rightarrow (\{0, 1\}^n)^k$ (where $l \ll nk$) and use $(C \circ f^{\otimes k}) \circ G : \{0, 1\}^l \rightarrow \{0, 1\}$, i.e.,

$$(C \circ f^{\otimes k}) \circ G(\sigma) = C(f(X_1), \dots, f(X_k)),$$

where $(X_1, \dots, X_k) \in (\{0, 1\}^n)^k$ is the output of $G(\sigma)$. This reduces the input length of the function to l . Therefore, if G is a “good” pseudorandom generator, we would expect $(C \circ f^{\otimes k}) \circ G$ to be harder (with respect to its input length) than $C \circ f^{\otimes k}$. We will show that this is indeed the case, provided that the generator G satisfies the following requirements:

1. *G is indistinguishability-preserving.* Analogously to Lemma 3.5, the generator G should be such that the computational hardness of $(C \circ f^{\otimes k}) \circ G$ is at least the information-theoretic hardness of $(C \circ g^{\otimes k}) \circ G$ for some δ -random function g —that is, at least $1/2 - \text{ExpBias}[(C \circ g^{\otimes k}) \circ G]$. We will see that this can be achieved, provided that G is *indistinguishability-preserving*; that is (analogously to the last part of Lemma 3.2),

$$\sigma \cdot f(X_1) \cdots f(X_k) \quad \text{and} \quad \sigma \cdot g(X_1) \cdots g(X_k)$$

should be indistinguishable, for some δ -random g , when $\sigma \stackrel{\text{R}}{\leftarrow} \{0, 1\}^l$ and $(X_1, \dots, X_k) \in (\{0, 1\}^n)^k$ is the output of G on input σ .

2. *G fools the expected bias.* G should be such that for any δ -random g $\text{ExpBias}[(C \circ g^{\otimes k}) \circ G]$ is approximately $\text{ExpBias}[C \circ g^{\otimes k}]$, and thus, by Lemma 3.7,

$$(2) \quad \text{ExpBias} [(C \circ g^{\otimes k}) \circ G] \leq \sqrt{\text{NoiseStab}_\delta[C] + \epsilon}$$

⁴A comment is in order about the input lengths for which f' is hard. As it turns out, the hardness of f' on inputs of length m is related to the hardness of the original function f on inputs of length $\Theta(\sqrt{m})$. Thus if f is hard for all sufficiently large input lengths, then so is f' . Alternatively, if f is hard only infinitely often, then we may still conclude that f' is hard infinitely often.

for a suitably small ϵ . Actually, we will not show that G fools the expected bias directly but instead will work with a related quantity (the expected collision probability), which will still suffice to show inequality (2).

Informally, the effect of the two above requirements on the generator G is that the hardness of $(C \circ f^{\otimes k}) \circ G$ is roughly the hardness of $C \circ f^{\otimes k}$, while the input length is dramatically reduced from nk to l (the seed length of G). More precisely, the first requirement allows us to relate the hardness of $(C \circ f^{\otimes k}) \circ G$ to the information-theoretic hardness of $(C \circ g^{\otimes k}) \circ G$ (where g is a δ -random function); the second allows us to relate this information-theoretic hardness to the noise stability of the combining function C . In particular, if we employ the combining function from Lemma 3.8, we obtain hardness $1/2 - 1/k^{\Omega(1)}$. Thus, by choosing $k \gg l$, we bypass the barrier discussed at the end of the previous section.

Now we briefly describe how the above requirements on G are met. The first requirement is achieved through a generator that outputs *combinatorial designs*. This construction is essentially from Nisan [28] and Nisan and Wigderson [30] and has been used in many places, e.g., [20, 33].

The second requirement is achieved as follows. We show that if G is pseudorandom against space-bounded algorithms and the combining function C is computable in small space (with one-way access to its input), then inequality (2) holds. We then use Nisan's *unconditional* pseudorandom generator against space-bounded algorithms [29], and show that combining functions with low noise stability can in fact be computed in small space.⁵ Note that we use the pseudorandomness of the generator G only to relate the expected bias with respect to G to a combinatorial property of the combining function C . In particular, it is *not* used to fool the circuits trying to compute the hard function. This is what allows us to use an unconditional generator against a relatively weak model of computation.

Our final generator, Γ , is the generator obtained by XORing a generator that is indistinguishability-preserving and a generator that fools the expected bias, yielding a generator that has both properties. The approach of XORing two generators in this way appeared in [20] and was subsequently used in [33].

4.2. Using nondeterminism. The derandomization described above gives hardness amplification up to $1/2 - 1/n^c$ for any constant c . This already improves upon the best previous result, namely Theorem 3.9. However, to go beyond this new techniques are required. The problem is that if we want C to be computable in time $\text{poly}(n)$, we must take $k = \text{poly}(n)$, and thus we amplify to at most $1/2 - 1/k = 1/2 - 1/\text{poly}(n)$.

We solve this problem by taking full advantage of the power of \mathcal{NP} , namely nondeterminism. This allows us to use a function $C : \{0, 1\}^k \rightarrow \{0, 1\}$ which is computable in *nondeterministic* time $\text{poly}(n, \log(k))$; thus, the amplified function will still be in \mathcal{NP} for k as large as 2^n .

Conversely, in section 7.2 we show that any nonadaptive monotone black-box hardness amplification that amplifies to hardness $1/2 - 1/n^{\omega(1)}$ cannot be computed in \mathcal{P} ; i.e., the use of nondeterminism is essential.

We proceed by discussing the details of the derandomization (sections 5.1, 5.2, and 5.3) and the use of nondeterminism (section 5.4). The results obtained in these sections are summarized in Table 1. For clarity of exposition, we focus on the case where the original hard function f is balanced and is $1/3$ -hard. Hardness amplification

⁵The same approach also works using the unconditional pseudorandom generator against constant-depth circuits of [28] and showing that the combining function is computable by a small constant-depth circuit; however, the space generator gives us slightly better parameters.

TABLE 1
Hardness amplification within \mathcal{NP} .

Functions : $\{0, 1\}^n \rightarrow \{0, 1\}$		
Amplification up to	Technique	Reference
$1/2 - 1/\sqrt{n}$	Direct product	[31]
$1/2 - 1/n^c$, for every c	Derandomized direct product	Theorem 5.8
$1/2 - 1/2^{\Omega(\sqrt{n})}$	Derandomized direct product & nondeterminism	Theorem 5.13

from hardness $1/\text{poly}(n)$ is discussed in section 5.5, and hardness amplification of unbalanced functions is discussed in section 7.1.

5. Proof of main theorem. In this section we prove our main theorem (i.e., Theorem 4.1).

5.1. Preserving indistinguishability. The main result in this subsection is that if G is pseudorandom in an appropriate sense, then the hardness of $(C \circ f^{\otimes k}) \circ G$ is roughly

$$1/2 - \text{ExpBias} [(C \circ g^{\otimes k}) \circ G]$$

for some δ -random function g . As we noted in the previous section, it will be sufficient for G to be *indistinguishability-preserving*. We give the definition of indistinguishability-preserving and then our main result.

DEFINITION 5.1. A generator $G : \{0, 1\}^l \rightarrow (\{0, 1\}^n)^k$ is said to be indistinguishability-preserving for size t if for all (possibly probabilistic) functions $f_1, \dots, f_k, g_1, \dots, g_k$ the following holds: If for every $i, 1 \leq i \leq k$, the distributions

$$U_n \cdot f_i(U_n) \quad \text{and} \quad U_n \cdot g_i(U_n)$$

are ϵ -indistinguishable for size s , then

$$\sigma \cdot f_1(X_1) \cdots f_k(X_k) \quad \text{and} \quad \sigma \cdot g_1(X_1) \cdots g_k(X_k)$$

are $k\epsilon$ -indistinguishable for size $s - t$, where σ is a random seed of length l and $X_1 \cdots X_k$ is the output of $G(\sigma)$.

The fact that in the above definition we consider k f_i 's and k g_i 's implies that an *indistinguishability-preserving* generator stays *indistinguishability-preserving* when XORed with any other generator (cf. the proof of item 1 in Lemma 5.12). We will use this property in the proof of our main result.

LEMMA 5.2. Let $f : \{0, 1\}^n \rightarrow \{0, 1\}$ be δ -hard for size s , let $G : \{0, 1\}^l \rightarrow (\{0, 1\}^n)^k$ be a generator that is *indistinguishability-preserving* for size t , and let $C : \{0, 1\}^k \rightarrow \{0, 1\}$ be any function. Then there exists a δ' -random g , with $\delta/2 \leq \delta' \leq \delta$ such that the function $(C \circ f^{\otimes k}) \circ G : \{0, 1\}^l \rightarrow \{0, 1\}$ has hardness

$$\frac{1}{2} - \frac{\text{ExpBias} [(C \circ g^{\otimes k}) \circ G]}{2} - \frac{k}{s^{1/3}}$$

for circuits of size $\Omega(s^{1/3}/\log(1/\delta)) - t - \text{size}(C)$, where $\text{size}(C)$ denotes the size of a smallest circuit computing C .

Proof. By Lemma 3.2, there exists a δ' -random function g with $\delta/2 \leq \delta' \leq \delta$ such that $U_n \cdot f(U_n)$ and $U_n \cdot g(U_n)$ are ϵ -indistinguishable for size $\Omega(s\epsilon^2/\log(1/\delta))$. Since G is indistinguishability-preserving for size t by assumption, this implies that

$$\sigma \cdot f(X_1) \cdots f(X_k) \quad \text{and} \quad \sigma \cdot g(X_1) \cdots g(X_k)$$

are $k\epsilon$ -indistinguishable for size $\Omega(s\epsilon^2/\log(1/\delta)) - t$, where here and below σ denotes a uniform random seed in $\{0, 1\}^l$ and $X_1 \cdots X_k$ will denote the output of $G(\sigma)$. This in turn implies that

$$\sigma \cdot C(f(X_1) \cdots f(X_k)) \quad \text{and} \quad \sigma \cdot C(g(X_1) \cdots g(X_k))$$

(i.e., $\sigma \cdot (C \circ f^{\otimes k}) \circ G(\sigma)$ and $\sigma \cdot (C \circ g^{\otimes k}) \circ G(\sigma)$) are $k\epsilon$ -indistinguishable for size $\Omega(s\epsilon^2/\log(1/\delta)) - t - \text{size}(C)$. By Lemma 3.4,

$$\sigma \cdot (C \circ g^{\otimes k}) \circ G \quad \text{and} \quad \sigma \cdot U_1$$

are $(\text{ExpBias}[(C \circ g^{\otimes k}) \circ G]/2)$ -indistinguishable for any size. Therefore, we have that

$$\sigma \cdot (C \circ f^{\otimes k}) \circ G \quad \text{and} \quad \sigma \cdot U_1$$

are $(\text{ExpBias}[(C \circ g^{\otimes k}) \circ G]/2 + k\epsilon)$ -indistinguishable for size $\Omega(s\epsilon^2/\log(1/\delta)) - t - \text{size}(C)$. The result follows by setting $\epsilon = 1/s^{1/3}$ and applying Lemma 2.3. \square

In particular, we note that the *identity generator* $G : \{0, 1\}^{nk} \rightarrow (\{0, 1\}^n)^k$, i.e., $G(x) = x$, is indistinguishability-preserving for size 0 (by a hybrid argument, see, e.g., [13]), and thus Lemma 3.5 is a corollary of Lemma 5.2. However, the identity generator has seed length nk and is therefore a very poor pseudorandom generator. Fortunately, there are indistinguishability-preserving pseudorandom generators with much shorter seeds which will allow us to use Lemma 5.2 to obtain much stronger hardness amplifications.

LEMMA 5.3. *There is a constant c such that for every $n \geq 2$ and every $k = k(n)$ there is an explicitly computable generator $NW_k : \{0, 1\}^l \rightarrow (\{0, 1\}^n)^k$ with seed length $l = c \cdot n^2$ that is indistinguishability-preserving for size k^2 .*

Proof. The generator is the main component of the generator by Nisan [28] and Nisan and Wigderson [30] and is based on combinatorial designs. Specifically, we let $S_1, \dots, S_k \subseteq [l]$ be an explicit family of sets such that $|S_i| = n$ for all i , and $|S_i \cap S_j| \leq \log k$ for all $i \neq j$. Nisan [28] gives an explicit construction of such sets with $l = O(n^2)$. Then the generator $NW_k : \{0, 1\}^l \rightarrow (\{0, 1\}^n)^k$ is defined by

$$NW_k(\sigma) := (\sigma|_{S_1}, \dots, \sigma|_{S_k}),$$

where $\sigma|_{S_i} \in \{0, 1\}^n$ denotes the projection of σ onto the coordinates indexed by the set S_i .

The proof that this generator is indistinguishability-preserving for size k^2 follows the arguments in [30, 33]. For completeness, we sketch the proof here. Suppose that we have a circuit C of size $s - k^2$ distinguishing the distributions $\sigma \cdot f_1(\sigma|_{S_1}) \cdots f_k(\sigma|_{S_k})$ and $\sigma \cdot g_1(\sigma|_{S_1}) \cdots g_k(\sigma|_{S_k})$ with advantage greater than $k \cdot \epsilon$. For $i = 0, \dots, k$, let H_i be the hybrid distribution

$$H_i = \sigma \cdot g_1(\sigma|_{S_1}) \cdots g_i(\sigma|_{S_i}) \cdot f_{i+1}(\sigma|_{S_{i+1}}) \cdots f_k(\sigma|_{S_k}).$$

Then there must exist an $i \in \{0, \dots, k\}$ such that C distinguishes H_i from H_{i+1} with advantage greater than ϵ . The only difference between H_i and H_{i+1} is that H_i has the component $f_{i+1}(\sigma|_{S_{i+1}})$, while H_{i+1} has $g_{i+1}(\sigma|_{S_{i+1}})$. By averaging, we may fix all the bits of σ outside of S_{i+1} (as well as the randomness of f_j, g_j for $j \neq i + 1$ if they are probabilistic functions) while preserving the advantage of C . Thus C distinguishes between two distributions of the form

$$\tau \cdot h_1(\tau)h_2(\tau) \cdots h_i(\tau)f_{i+1}(\tau)h_{i+2}(\tau) \cdots h_k(\tau)$$

and

$$\tau \cdot h_1(\tau)h_2(\tau) \cdots h_i(\tau)g_{i+1}(\tau)h_{i+2}(\tau) \cdots h_k(\tau),$$

where τ is uniform in $\{0, 1\}^n$ and each h_j is a function of at most $|S_j \cap S_{i+1}|$ bits of τ . Then each h_j can be computed by a circuit of size smaller than $2^{|S_j \cap S_{i+1}|} \leq k$. Combining these $k - 1$ circuits with C , we get a distinguisher between $\tau \cdot f_{i+1}(\tau)$ and $\tau \cdot g_{i+1}(\tau)$ of size $|C| + (k - 1) \cdot k < s$ and advantage greater than ϵ . \square

5.2. Fooling the expected bias. In this subsection we prove a derandomized version of Lemma 3.7. Informally, we show that if C is computable in a restricted model of computation and G “fools” that restricted model of computation, then for any δ -random function g ,

$$\text{ExpBias} [(C \circ g^{\otimes k}) \circ G] \leq \sqrt{\text{NoiseStab}_\delta[C] + \epsilon}.$$

The restricted model of computation we consider is that of nonuniform space-bounded algorithms that make one pass through the input, reading it in blocks of length n . These are formally modeled by the following kind of branching programs.

DEFINITION 5.4. *A (probabilistic, read-once, oblivious) branching program of size s with block-size n is a finite state machine with s states, over the alphabet $\{0, 1\}^n$ (with a fixed start state and an arbitrary number of accepting states). Each edge is labeled with a symbol in $\{0, 1\}^n$. For every state a and symbol $\alpha \in \{0, 1\}^n$, the edges leaving a and labeled with α are assigned a probability distribution. Then computation proceeds as follows. The input is read sequentially, one block of n bits at a time. If the machine is in state a and it reads α , then it chooses an edge leaving a and labeled with α according to its probability, and moves along it.*

Now we formally define pseudorandom generators against branching programs.

DEFINITION 5.5. *A generator $G : \{0, 1\}^l \rightarrow (\{0, 1\}^n)^k$ is ϵ -pseudorandom against branching programs of size s and block-size n if for every branching program B of size s and block-size n :*

$$|\Pr[B(G(U_l)) = 1] - \Pr[B(U_{nk}) = 1]| \leq \epsilon.$$

In [29], Nisan builds an unconditional pseudorandom generator against branching programs. Its parameters (specialized for our purposes) are given in the following theorem.

THEOREM 5.6 (see [29]). *For every n and $k \leq 2^n$, there exists a generator*

$$N_k : \{0, 1\}^l \rightarrow (\{0, 1\}^n)^k$$

such that

- N_k is 2^{-n} -pseudorandom against branching programs of size 2^n and block-size n ,

- N_k has seed length $l = O(n \log k)$,
- N_k is explicitly computable.

Note that Nisan [29] does not mention *probabilistic* branching programs. However, if there is a probabilistic branching program distinguishing the output of the generator from uniform, then by a fixing of the coin tosses of the branching program there is a *deterministic* branching program that distinguishes the output of the generator from uniform.

We now state the derandomized version of Lemma 3.7.

LEMMA 5.7. *Let*

1. $g : \{0, 1\}^n \rightarrow \{0, 1\}$ be a δ -random function,
2. $C : \{0, 1\}^k \rightarrow \{0, 1\}$ be computable by a branching program of size t and block-size 1,
3. $G : \{0, 1\}^l \rightarrow (\{0, 1\}^n)^k$ be $\epsilon/2$ -pseudorandom against branching programs of size t^2 and block-size n .

Then $\text{ExpBias}[(C \circ g^{\otimes k}) \circ G] \leq \sqrt{\text{NoiseStab}_\delta[C] + \epsilon}$.

Proof. We will not show that G fools the expected bias, but rather the following related quantity. For a probabilistic Boolean function $h(x; r)$ we define its (normalized) *expected collision probability* as

$$\text{ExpCP}[h] \stackrel{\text{def}}{=} \mathbb{E}_x \left[2 \cdot \Pr_{r, r'} [h(x; r) = h(x; r')] - 1 \right].$$

The same reasoning that proves Lemma 3.7 also shows that for every probabilistic Boolean function h ,

$$(3) \quad \text{ExpBias}[h] \leq \sqrt{\text{ExpCP}[h]}.$$

More specifically, inequality (3) holds because

$$\begin{aligned} \text{ExpBias}[h] &= \mathbb{E}_{x \leftarrow U_n} [\text{Bias}[h(x)]] \\ &\leq \sqrt{\mathbb{E}_{x \leftarrow U_n} [\text{Bias}[h(x)]^2]} \quad (\text{by Cauchy-Schwarz}) \\ &= \sqrt{\text{ExpCP}[h]}. \end{aligned}$$

Let $h(x; r) : (\{0, 1\}^n)^k \rightarrow \{0, 1\}$ be the probabilistic function $C \circ g^{\otimes k}$. Even though h is defined in terms of g , it turns out that its expected collision probability is the same for all δ -random functions g . Specifically, for $x = (x_1, \dots, x_k)$, the only dependence of the collision probability $\Pr_{r, r'} [h(x; r) = h(x; r')]$ on x_i comes from whether $g(x_i)$ is a coin flip (which occurs with probability δ over the choice of x_i), $g(x_i) = 1$ (which occurs with probability $(1 - \delta)/2$), or $g(x_i) = 0$ (which occurs with probability $(1 - \delta)/2$). In the case where $g(x_i)$ is a coin flip, then the i th bits of the two inputs fed to C (i.e., $g(x_i; r)$ and $g(x_i; r')$) are random and independent, and otherwise they are equal and fixed (according to $g(x_i)$). It can be verified that this corresponds precisely to the definition of noise stability, so that we have

$$(4) \quad \text{ExpCP}[h] = \text{NoiseStab}_\delta[C].$$

Now we construct a probabilistic branching program $M : (\{0, 1\}^n)^k \rightarrow \{0, 1\}$ of size t^2 and block-size n such that for every $x \in (\{0, 1\}^n)^k$:

$$\Pr[M(x) = 1] = \Pr_{r, r'} [h(x; r) = h(x; r')].$$

To do this, we first note that, using the branching program for C , we can build a probabilistic branching program of size t with block-size n which computes $C \circ g^{\otimes k}$: The states of the branching program are the same as those of the branching program for C , and we define the transitions as follows. Upon reading symbol $\alpha \in \{0, 1\}^n$ in state s , if $g(\alpha) = 0$ (resp. $g(\alpha) = 1$), we deterministically go to the state given by the 0-transition (resp., 1-transition) of C from state s , and if $g(\alpha)$ is a coin flip, then we put equal probability on these two transitions.

Then, to obtain M , run two *independent* copies of this branching program (i.e., using independent choices for the probabilistic state transitions) and accept if and only if both of them accept or both of them reject. Now,

$$\begin{aligned} & \left| \text{ExpCP}[(C \circ g^{\otimes k}) \circ G] - \text{NoiseStab}_\delta[C] \right| \\ &= \left| \text{ExpCP}[(C \circ g^{\otimes k}) \circ G] - \text{ExpCP}[C \circ g^{\otimes k}] \right| \quad (\text{by (4)}) \\ &= 2 \cdot \left| \Pr[M \circ G(U_l) = 1] - \Pr[M(U_{n \cdot k}) = 1] \right| \\ &\leq \epsilon. \quad (\text{by pseudorandomness of } G) \end{aligned}$$

The lemma follows, combining this with (3). \square

5.3. Amplification up to $1/2 - 1/\text{poly}$. In this subsection we sketch our hardness amplification up to $1/2 - 1/n^c$, for every c , as follows.

THEOREM 5.8. *If there is a balanced function $f : \{0, 1\}^n \rightarrow \{0, 1\}$ in \mathcal{NP} that is $(1/3)$ -hard for size $s(n) \geq n^{\omega(1)}$, then for every $c > 0$ there is a function $f' : \{0, 1\}^m \rightarrow \{0, 1\}$ in \mathcal{NP} that is $(1/2 - 1/m^c)$ -hard for size $(s(\sqrt{m}))^{\Omega(1)}$.*

To amplify we use the Tribes function of Ben-Or and Linial [5], a monotone read-once DNF.

DEFINITION 5.9. *The Tribes function on k bits is*

$$\begin{aligned} \text{Tribes}_k(x_1, \dots, x_k) &\stackrel{\text{def}}{=} \\ &(x_1 \wedge \dots \wedge x_b) \vee (x_{b+1} \wedge \dots \wedge x_{2b}) \vee \dots \vee (x_{k-b+1} \wedge \dots \wedge x_k) \end{aligned}$$

where there are k/b clauses each of size b , and b is the largest integer such that $(1 - 2^{-b})^{k/b} \geq 1/2$. Note that this makes $b = O(\log k)$.

The Tribes DNF has very low noise stability when perturbed with constant noise.

LEMMA 5.10 (see [31, 27]). *For every constant $\delta > 0$,*

$$\text{NoiseStab}_\delta[\text{Tribes}_k] \leq \frac{1}{k^{\Omega(1)}}.$$

A key step in our result is that Tribes_k is easily computable by a branching program of size $O(k)$, and therefore we can use Lemma 5.7 to fool its expected bias.

We now define the generator that we will use in our derandomized direct product construction.

DEFINITION 5.11. *Given n and $k \leq 2^n$, define the generator $\Gamma_k : \{0, 1\}^m \rightarrow (\{0, 1\}^n)^k$ as follows:*

$$\Gamma_k(x, y) \stackrel{\text{def}}{=} NW_k(x) \oplus N_k(y),$$

where \oplus denotes bitwise XOR.

We recall the properties of Γ that we are interested in, as follows.

LEMMA 5.12. *The following hold:*

1. Γ_k is indistinguishability-preserving for size k^2 .
2. Γ_k is 2^{-n} -pseudorandom against branching programs of size 2^n and block-size n .
3. Γ_k has seed length $m = O(n^2)$.
4. Γ_k is explicitly computable (see Definition 4.2 for the definition of explicit).

Proof. . Item 1 follows from Lemma 5.3 and the fact that an indistinguishability-preserving generator XORed with any fixed string is still indistinguishability-preserving. More specifically, suppose, for the sake of contradiction, that $\Gamma_k(x, y) = NW_k(x) \oplus N_k(y)$ is not indistinguishability-preserving. Then there are functions f_1, \dots, f_k and g_1, \dots, g_k such that for every i the distributions $U_n \cdot f_i(U_n)$ and $U_n \cdot g_i(U_n)$ are indistinguishable, yet the distributions

$$(x, y) \cdot f_1(NW_1(x) \oplus N_1(y)) \cdots f_k(NW_k(x) \oplus N_k(y))$$

and

$$(x, y) \cdot g_1(NW_1(x) \oplus N_1(y)) \cdots g_k(NW_k(x) \oplus N_k(y))$$

are distinguishable (for random x, y). Then, by averaging, they are distinguishable for some fixed value of $y = \tilde{y}$. Thus, we obtain that

$$x \cdot f'_1(NW_1(x)) \cdots f'_k(NW_k(x))$$

and

$$x \cdot g'_1(NW_1(x)) \cdots g'_k(NW_k(x))$$

are distinguishable (for random x), where $f'_i(z) = f_i(z \oplus N_i(\tilde{y}))$, $g'_i(z) = g_i(z \oplus N_i(\tilde{y}))$. (Note that we hardwire the fixed part of the seed \tilde{y} in the distinguisher.) Now observe that the indistinguishability of $U_n \cdot f_i(U_n)$ and $U_n \cdot g_i(U_n)$ implies the indistinguishability of $U_n \cdot f'_i(U_n)$ and $U_n \cdot g'_i(U_n)$, because the mapping $T(u \cdot v) = (u \oplus N_i(\tilde{y})) \cdot v$ transforms the latter pair of distributions to the former. (There is no loss in the circuit size, assuming that circuits have input gates for both the input variables and their negations.) But this is a contradiction because NW is indistinguishability-preserving.

Item 2 follows from Theorem 5.6 and the fact that XORing with any fixed string (in particular, $NW_k(x)$ for any x) preserves pseudorandomness against branching programs.

Item 3 is an immediate consequence of the seed lengths of NW_k (Lemma 5.3) and N_k (Theorem 5.6).

Item 4 follows from the facts that NW_k is explicit (Lemma 5.3) and N_k is explicit (Theorem 5.6). \square

Proof of Theorem 5.8. Given $f : \{0, 1\}^n \rightarrow \{0, 1\}$ that is δ -hard for size $s(n)$ (for $\delta = 1/3$) and a constant c , let $k = n^{c'}$ for $c' = O(c)$ to be determined later. Consider the function $f' : \{0, 1\}^m \rightarrow \{0, 1\}$ defined by

$$f' \stackrel{\text{def}}{=} (\text{Tribes}_k \circ f^{\otimes k}) \circ \Gamma_k.$$

Note that $f' \in \mathcal{NP}$ since $f \in \mathcal{NP}$, Tribes is monotone, and both Γ and Tribes are efficiently computable.

We now analyze the hardness of f' . Since Γ_k is indistinguishability-preserving for size k^2 by Lemma 5.12, Lemma 5.2 implies that there is a δ' -random function g (for $\delta/2 \leq \delta' \leq \delta$) such that f' has hardness

$$(5) \quad \frac{1}{2} - \frac{\text{ExpBias}[(\text{Tribes}_k \circ g^{\otimes k}) \circ \Gamma_k]}{2} - \frac{k}{s(n)^{1/3}}$$

for circuits of size $\Omega(s(n)^{1/3}) - k^2 - \text{size}(\text{Tribes}_k)$. Next we bound the hardness. By Lemma 5.12, we know that Γ_k is 2^{-n} -pseudorandom against branching programs of size 2^n and block-size n . In particular, since $k = \text{poly}(n)$, Γ_k is $1/k$ -pseudorandom against branching programs of size $9k$ and block-size n . Since, as we noted before, Tribes_k is easily computable by a branching program of size $O(k)$, we can apply Lemma 5.7 (noting that $O(k)^2 = \text{poly}(n) \ll 2^n$) in order to bound $\text{ExpBias}[(\text{Tribes}_k \circ g^{\otimes k}) \circ \Gamma_k]$ by $\sqrt{\text{NoiseStab}_{\delta'}[\text{Tribes}_k] + 2/k}$. And the noise stability inside the square root is at most $1/k^{\Omega(1)}$ by Lemma 5.10. Since $k = \text{poly}(n)$ and $s(n) = n^{\omega(1)}$, the $k/s^{1/3}$ term in the hardness (5) is negligible, and we obtain hardness at least $1/2 - 1/k^{\Omega(1)}$.

We now bound the circuit size: Since Tribes_k is computable by circuits of size $O(k)$, and since $s(n) = n^{\omega(1)}$, the size is at least $s(n)^{\Omega(1)}$.

Now note that f' has input length $m = m(n) = O(n^2)$ by Lemma 5.12. Strictly speaking, we have defined f' only for certain input lengths; however, it is easy to extend the function to every input length simply by defining $f'(x) \stackrel{\text{def}}{=} f'(x')$, where x' consists of the first $m(n)$ bits of x and n is the largest integer such that $m(n) \leq |x|$. It is easy to check that f' still has hardness $1/2 - 1/k^{\Omega(1)} = 1/2 - 1/n^{\Omega(c')}$. The result then follows by an appropriate choice of $c' = O(c)$. \square

5.4. Using nondeterminism. In this subsection we discuss how to use nondeterminism to get the following theorem.

THEOREM 5.13. *If there is a balanced function $f : \{0, 1\}^n \rightarrow \{0, 1\}$ in \mathcal{NP} that is $(1/3)$ -hard for size $s(n)$, then there is a function $f' : \{0, 1\}^m \rightarrow \{0, 1\}$ in \mathcal{NP} that is $(1/2 - 1/s(\sqrt{m})^{\Omega(1)})$ -hard for size $s(\sqrt{m})^{\Omega(1)}$.*

Our main observation is that Tribes_k is a DNF with clause size $O(\log k)$, and therefore it is computable in nondeterministic time $\text{poly}(n)$ even when k is superpolynomial in n .

LEMMA 5.14. *Let $f : \{0, 1\}^n \rightarrow \{0, 1\}$ be in \mathcal{NP} , and let $G_k : \{0, 1\}^l \rightarrow (\{0, 1\}^n)^k$ be any explicitly computable generator (see Definition 4.2) with $l \geq n$. Then the function $f' \stackrel{\text{def}}{=} (\text{Tribes}_k \circ f^{\otimes k}) \circ G_k$ is computable in \mathcal{NP} for every $k = k(n) \leq 2^n$.*

Proof. We compute $f'(\sigma)$ nondeterministically as follows. Guess a clause $v_i \wedge v_{i+1} \wedge \dots \wedge v_j$ in Tribes_k . Accept if for every h such that $i \leq h \leq j$ we have $f(X_h) = 1$, where $G(\sigma) = (X_1, \dots, X_k)$ and the values $f(X_h)$ are computed using the \mathcal{NP} algorithm for f .

It can be verified that this algorithm has an accepting computation path on input σ iff $f'(\sigma) = 1$. Note that the clauses have size logarithmic in k , which is polynomial in n . Moreover, G is explicitly computable. The result follows. \square

Now the proof of Theorem 5.13 proceeds along the same lines as the proof of Theorem 5.8, setting $k \stackrel{\text{def}}{=} s(n)^{\Omega(1)}$.

5.5. Amplifying from hardness $1/\text{poly}$. Our amplification from hardness $\Omega(1)$ to $1/2 - \epsilon$ (Theorem 5.8) can be combined with O'Donnell's amplification from hardness $1/\text{poly}$ to hardness $\Omega(1)$ to obtain an amplification from $1/\text{poly}$ to $1/2 - \epsilon$.

However, since O’Donnell’s construction blows up the input length polynomially, we would obtain only $\epsilon = 1/s(n^{\Omega(1)})$ (where the hidden constant depends on the initial polynomial hardness) rather than $\epsilon = 1/s(\sqrt{n})^{\Omega(1)}$ (as in Theorem 5.8). Thus we show here how to amplify directly from $1/\text{poly}$ to $1/2 - \epsilon$ using our approach. For this we need a combining function C that is more involved than the Tribes function. The properties of C that are needed in the proof of Theorem 4.1 are captured by the following lemma.

LEMMA 5.15. *For every $\delta(n) = 1/n^{O(1)}$, there is a sequence of functions $C_k : \{0, 1\}^k \rightarrow \{0, 1\}$ such that for every $k = k(n)$ with $n^{\omega(1)} \leq k \leq 2^n$ the following hold:*

1. $\text{NoiseStab}_\delta[C_k] \leq 1/k^{\Omega(1)}$.
2. For every $f : \{0, 1\}^n \rightarrow \{0, 1\}$ in \mathcal{NP} and every explicitly computable generator (see Definition 4.2) $G_k : \{0, 1\}^l \rightarrow (\{0, 1\}^n)^k$ with $l \geq n$, the function $(C_k \circ f^{\otimes k}) \circ G_k$ is in \mathcal{NP} .
3. C_k can be computed by a branching program of size $\text{poly}(n) \cdot k$ and also by a circuit of size $\text{poly}(n) \cdot k$.

Before proving Lemma 5.15, let us see how it can be used to prove our main theorem.

THEOREM 5.16 (Theorem 4.1, restated). *If there is a balanced function $f : \{0, 1\}^n \rightarrow \{0, 1\}$ in \mathcal{NP} that is $1/\text{poly}(n)$ -hard for size $s(n)$, then there is a function $f' : \{0, 1\}^m \rightarrow \{0, 1\}$ in \mathcal{NP} that is $(1/2 - 1/s(\sqrt{m})^{\Omega(1)})$ -hard for size $s(\sqrt{m})^{\Omega(1)}$.*

Proof. Let $f : \{0, 1\}^n \rightarrow \{0, 1\}$ be a balanced function in \mathcal{NP} that is $\delta = \delta(n)$ -hard for size $s(n)$, where $\delta \geq 1/n^{O(1)}$. Let $k = k(n) \stackrel{\text{def}}{=} s(n)^{1/7}$, and let C_k be the function guaranteed by Lemma 5.15. Let Γ_k be the generator from Definition 5.11. Consider the function $f' : \{0, 1\}^m \rightarrow \{0, 1\}$ defined by $f' \stackrel{\text{def}}{=} (C_k \circ f^{\otimes k}) \circ \Gamma_k$. Note that $f' \in \mathcal{NP}$, by item 2 in Lemma 5.15.

We now analyze the hardness of f' . Since Γ_k is indistinguishability-preserving for size k^2 (by Lemma 5.12), Lemma 5.2 implies that there is a δ' -random function g (for $\delta/2 \leq \delta' \leq \delta$) such that f' has hardness

$$(6) \quad \alpha(m) = \frac{1}{2} - \frac{\text{ExpBias}[(C_k \circ g^{\otimes k}) \circ \Gamma_k]}{2} - \frac{k}{s(n)^{1/3}}$$

for circuits of size

$$s'(m) = \Omega\left(\frac{s(n)^{1/3}}{\log(1/\delta)}\right) - k^2 - \text{size}(C_k).$$

We first bound the hardness $\alpha(m)$. By Lemma 5.12, we know that Γ_k is 2^{-n} -pseudorandom against branching programs of size 2^n and block-size n . Since the branching program for computing C_k has size $\text{poly}(n) \cdot k$, and $(\text{poly}(n) \cdot k)^2 \ll 2^n$ (by our choice of $k(n)$), we may apply Lemma 5.7 in order to bound $\text{ExpBias}[(C_k \circ g^{\otimes k}) \circ \Gamma_k]$ by $\sqrt{\text{NoiseStab}_{\delta'}[C_k]} + 2/2^n$. This noise stability is at most $1/k^{\Omega(1)}$ by item 1 in Lemma 5.15. Using the fact that $k = s(n)^{1/7}$, we have

$$\alpha(m) \geq \frac{1}{2} - \frac{\sqrt{1/k^{\Omega(1)} - 2/2^n}}{2} - \frac{k}{s(n)^{1/3}} = \frac{1}{2} - \frac{1}{s(n)^{\Omega(1)}}.$$

We now bound the circuit size $s'(m)$. Since C_k is computable by a circuit of size $\text{poly}(n) \cdot k$ (by item 3 in Lemma 5.15) and $\log(1/\delta) = O(\log n)$ and $s(n) = n^{\omega(1)}$, we have

$$s'(m) = \Omega\left(\frac{s(n)^{1/3}}{\log n}\right) - s(n)^{2/7} - \text{poly}(n) = s(n)^{\Omega(1)}.$$

To conclude, we note that f' has input length $m = O(n^2)$ by Lemma 5.12, so $s(n) = s(\Omega(\sqrt{m})) = s(\sqrt{m})^{\Omega(1)}$, and we indeed obtain hardness $\alpha(m) = 1/2 - 1/s(\sqrt{m})^{\Omega(1)}$ for size $s'(m) = s(\sqrt{m})^{\Omega(1)}$. Strictly speaking, we have defined f' only for certain input lengths; however, it is easy to extend the function to every input length; cf. the end of the proof of Theorem 5.8. \square

The rest of this subsection is devoted to the proof of Lemma 5.15. Recall that amplification from hardness $\Omega(1)$ (Theorem 5.8) relies on the fact that the Tribes DNF has low noise stability with respect to noise parameter $\delta = \Omega(1)$ (i.e., Lemma 5.10). Similarly, to amplify from hardness $1/\text{poly}(n)$ we need to employ a combining function that has low noise stability with respect to noise $1/\text{poly}(n)$. To this end, following [31], we employ the recursive-majorities function, RMaj_r . Let Maj denote the majority function.

DEFINITION 5.17. *The RMaj_r function on 3^r bits is defined recursively by*

$$\begin{aligned} \text{RMaj}_1(x_1, x_2, x_3) &\stackrel{\text{def}}{=} \text{Maj}(x_1, x_2, x_3), \\ \text{RMaj}_r(x_1, \dots, x_{3^r}) &\stackrel{\text{def}}{=} \text{RMaj}_{r-1}(\text{Maj}(x_1, x_2, x_3), \dots, \text{Maj}(x_{3^{r-2}}, x_{3^{r-1}}, x_{3^r})). \end{aligned}$$

The following lemma quantifies the noise stability of RMaj_r .

LEMMA 5.18 (see [31, Proposition 11]). *There is a constant c such that for every $\delta > 0$ and every $r \geq c \cdot \log(1/\delta)$ we have*

$$\text{NoiseStab}_\delta[\text{RMaj}_r] \leq \frac{1}{4}.$$

Note that if $r = O(\log n)$, then RMaj_r is a function of $3^r = \text{poly}(n)$ bits.

However, when $r = O(\log n)$, RMaj_r does not have sufficiently low noise stability to be used on its own. For this reason, we will combine RMaj with Tribes. (The same combination of RMaj and Tribes is employed by O'Donnell [31], albeit for a different setting of parameters.)

Proof of Lemma 5.15. Given n and $\delta = \delta(n) \geq 1/n^{O(1)}$, let $r \stackrel{\text{def}}{=} c \cdot \log(1/\delta)$ for a constant c to be chosen later. Assume, without loss of generality, that r and $k/3^r$ are integers. The function $C_k : \{0, 1\}^k \rightarrow \{0, 1\}$ is defined as

$$C_k \stackrel{\text{def}}{=} \text{Tribes}_{k/3^r} \circ \text{RMaj}_r^{\otimes k/3^r}.$$

We now prove that C_k satisfies the required properties.

Item 1. We will use the following result from [31].

LEMMA 5.19 (see [31, Proposition 8]). *If h is a balanced Boolean function and $\varphi : \{0, 1\}^\ell \rightarrow \{0, 1\}$ is any Boolean function, then*

$$\text{NoiseStab}_\delta[\varphi \circ h^{\otimes \ell}] = \text{NoiseStab}_{\frac{1}{2} - \frac{\text{NoiseStab}_\delta[h]}{2}}[\varphi].$$

Letting c be a sufficiently large constant (recall that $r = c \cdot \log(1/\delta)$), by Lemma 5.18 we have that $\text{NoiseStab}_\delta[\text{RMaj}_r]/2 \geq 1/2 - 1/8 \geq 3/8$. Now note that RMaj_r is balanced because taking the bitwise complement of an input x also negates the value of $\text{RMaj}_r(x)$. Hence, by Lemma 5.19,

$$\begin{aligned} \text{NoiseStab}_\delta[\text{Tribes}_{k/3^r} \circ \text{RMaj}_r^{\otimes k/3^r}] &= \text{NoiseStab}_{3/8}[\text{Tribes}_{k/3^r}] \\ &\leq \frac{1}{(k/3^r)^{\Omega(1)}} = \frac{1}{k^{\Omega(1)}}, \end{aligned}$$

where the last two equalities use Lemma 5.10 and the fact that $k = n^{\omega(1)}$ and $r = O(\log n)$.

The proof of item 2 is similar to the proof of Lemma 5.14. To compute $(\text{Tribes}_{k/3^r} \circ \text{RMaj}_r^{\otimes k/3^r} \circ f^{\otimes k}) \circ G_k$, we guess a clause of the $\text{Tribes}_{k/3^r}$ and verify that all the RMaj_r evaluations feeding into it are satisfied (using the \mathcal{NP} algorithm for f). The only additional observation is that each of the recursive majorities depends on only $3^r = \text{poly}(n)$ bits of the input and hence can be computed in time polynomial in n .

Item 3. As noted earlier, $\text{Tribes}_{k/3^r}$ is easily computable by a branching program of size $O(k)$. RMaj_r , on the other hand, can be computed by a branching program of size $\text{poly}(n)$. Indeed, $\text{Maj}(x_1, x_2, x_3)$ is clearly computable by a branching program of constant size c , and therefore $\text{RMaj}_r = \text{RMaj}_{r-1}(\text{Maj}(x_1, x_2, x_3), \dots, \text{Maj}(x_{3^{r-2}}, x_{3^{r-1}}, x_{3^r}))$ can be computed by a branching program whose size is at most c times the size of RMaj_{r-1} . By induction it follows that RMaj_r can be computed in size $c^r = \text{poly}(n)$.

By composing the branching program of size $O(k)$ for Tribes with the branching program of size $\text{poly}(n)$ for RMaj , we can compute C_k by a branching program of size $\text{poly}(n) \cdot k$. \square

A natural question is whether one can amplify from hardness $1/n^{\omega(1)}$. A modification of the [37] negative result about hardness amplification given in [26] shows that this task cannot be achieved by any black-box hardness amplification computable in the polynomial-time hierarchy (and thus in particular cannot be achieved by any black-box hardness amplification computable in \mathcal{NP}). (See Definition 7.2 for the definition of black-box hardness amplification.)

6. Extensions.

6.1. On the possibility of amplifying hardness up to $1/2 - 1/2^{\Omega(n)}$. Even when starting from a function that is δ -hard for size $2^{\Omega(n)}$, our results (Theorem 4.1) amplify hardness only up to $1/2 - 1/2^{\Omega(\sqrt{n})}$ (rather than $1/2 - 1/2^{\Omega(n)}$). In this section we discuss the possibility of amplifying hardness in \mathcal{NP} up to $1/2 - 1/2^{\Omega(n)}$ when starting with a function that is δ -hard for size $2^{\Omega(n)}$. The problem is that the seed length of our generator in Lemma 5.12 is *quadratic* in n , rather than linear. To amplify hardness up $1/2 - 1/2^{\Omega(n)}$ we need a generator (for every $k = 2^{\Omega(n)}$) with the same properties of the one in Lemma 5.12, but with linear seed length.

Recall that our generator is the XOR of an indistinguishability-preserving generator and a generator that is pseudorandom against branching programs. While it is an open problem to exhibit a generator with linear seed length that is pseudorandom against branching programs, an indistinguishability-preserving generator with linear seed length is given by the following lemma.

LEMMA 6.1. *For every constant γ , $0 < \gamma < 1$, there is a constant c such that for every n there is an explicitly computable generator $NW'_{2^{n/c}} : \{0, 1\}^l \rightarrow (\{0, 1\}^n)^{2^{n/c}}$ with seed length $l = c \cdot n$ that is indistinguishability-preserving for size $2^{\gamma \cdot n}$.*

The generator is due to Nisan and Wigderson [30] and Impagliazzo and Wigderson [20]. The approach is the same as for the generator used in Lemma 5.3, except we now require a design consisting of $2^{\Omega(n)}$ sets of size n in a universe of size $O(n)$, with pairwise intersections of size at most $\gamma n/2$. An explicit construction of such a design is given in [16].⁶

⁶Alternatively, we can use the randomized algorithm described in [20], which computes such sets S_1, \dots, S_M with probability exponentially close to 1 using $O(n)$ random bits. This is sufficient for constructing an indistinguishability-preserving generator.

THEOREM 6.2. *Suppose that there exists an explicit generator $N'_{2^n} : \{0, 1\}^l \rightarrow (\{0, 1\}^n)^{2^n}$ that is 2^{-n} -pseudorandom against branching programs of size 2^n and block-size n and that has seed length $l = O(n)$. Then the following holds: If there is a balanced function $f : \{0, 1\}^n \rightarrow \{0, 1\}$ in \mathcal{NP} that is $1/\text{poly}(n)$ -hard for size $2^{\Omega(n)}$, then there is a function $f' : \{0, 1\}^m \rightarrow \{0, 1\}$ in \mathcal{NP} that is $(1/2 - 1/2^{\Omega(n)})$ -hard for size $2^{\Omega(n)}$.*

A slightly more careful analysis shows that all the branching programs considered in our constructions have *width*⁷ much smaller than their size (specifically, the branching program for Tribes has constant width, and the one for the function in Lemma 5.15, i.e., Tribes composed with RMaj, has width $\text{poly}(n)$ independent of k). Thus to prove the conclusion in the statement of Theorem 6.2 it would suffice to exhibit an explicit pseudorandom generator that fools such restricted branching programs.

For amplifying from constant hardness, it suffices instead to have a generator fooling *constant-depth circuits* of size 2^n with seed length $O(n)$. (The generator of Nisan [28] has seed length $\text{poly}(n)$.) The reason is that our proof that generators against branching programs “fool” the expected bias also works for generators against constant-depth circuits, provided that the combining function is computable in constant depth. The Tribes function is depth 2 by definition (but the recursive majorities RMaj is not constant-depth, and hence this would only amplify from constant hardness).

More generally, we only need, for every constant $\gamma > 0$, a generator $G : \{0, 1\}^{O(n)} \rightarrow (\{0, 1\}^n)^k$, where $k = 2^{\gamma n}$, such that for every δ -random function g ,

$$\text{ExpBias} [(C_k \circ g^{\otimes k}) \circ G] = 2^{-\Omega(n)},$$

where, for example, $C_k = \text{Tribes}_k$ (when δ is constant). As in the proof of Lemma 3.7, in proving such a statement it may be convenient to work instead with the (polynomially related) expected collision probability. An important property of $C_k = \text{Tribes}_k$ used in bounding the expected bias with respect to G is that it gives expected bias $2^{-\Omega(n)}$ if G is replaced with a truly random generator (i.e., using seed length $n \cdot k$) and δ is constant. One might try to use a different monotone combining function with this property, provided that it can also be evaluated in nondeterministic time $\text{poly}(n)$.

6.2. Impagliazzo and Wigderson’s hardness amplification. Our framework gives a new proof of the hardness amplification by Impagliazzo and Wigderson [20]. Impagliazzo and Wigderson showed that if $\mathcal{E} \stackrel{\text{def}}{=} \text{DTIME}(2^{O(n)})$ contains a function $f : \{0, 1\}^n \rightarrow \{0, 1\}$ that requires (in the worst-case) circuits of size $2^{\Omega(n)}$, then \mathcal{E} contains a function f' that is $(1/2 - 1/2^{\Omega(n)})$ -hard for circuits of size $2^{\Omega(n)}$. The main contribution in [20] is amplification from constant hardness, e.g., $1/3$, to $(1/2 - 1/2^{\Omega(n)})$ (amplification from worst-case hardness and constant hardness was essentially already established in [3, 18]). The improvement over the standard Yao XOR lemma is that the input length of the amplified function increases by only a constant factor. In this section, we sketch a simple proof of this result using the framework developed in earlier sections. While other, more recent hardness amplifications achieving the same result for \mathcal{E} are known [33, 32, 36], the original one by Impagliazzo and Wigderson is still interesting because it can be implemented in “low” complexity classes, such as the polynomial-time hierarchy, while the others cannot (due to the fact that they actually amplify from worst-case hardness [37]).

⁷The width of a branching program is the maximum, over i , of the number of states that are reachable after reading i symbols.

The construction of [20] uses an *expander-walk* generator $W_k : \{0, 1\}^l \rightarrow (\{0, 1\}^n)^k$, which uses its seed of length $l = n + O(k)$ to do a random walk of length k (started at a random vertex) in a constant-degree expander graph on 2^n vertices. More background on such generators can be found in [12, section 3.6.3]. The construction of [20] XORs the expander-walk generator with the (first k outputs of the) indistinguishability-preserving generator from Lemma 6.1, as follows.

DEFINITION 6.3. *Let $k = c \cdot n$ for a constant $c > 1$. Let $NW''_k : \{0, 1\}^{O(n)} \rightarrow (\{0, 1\}^n)^k$ be a generator that is indistinguishability-preserving for size $2^{n/c}$, as given by Lemma 6.1. The generator $IW_k : \{0, 1\}^l \rightarrow (\{0, 1\}^n)^k$ is defined as*

$$IW_k(x, y) \stackrel{\text{def}}{=} NW''_k(x) \oplus W_k(y).$$

The seed length of IW_k is $l = O(n)$.

Given a function f that is $1/3$ -hard for size $s = 2^{\Omega(n)}$, the Impagliazzo–Wigderson amplification defines

$$f' \stackrel{\text{def}}{=} (XOR \circ f^{\otimes k}) \circ IW_k : \{0, 1\}^{O(n)} \rightarrow \{0, 1\},$$

where $k = c \cdot n$ for a constant c that depends on the hidden constant in the $s = 2^{\Omega(n)}$. They prove the following about this construction.

THEOREM 6.4 (after [20]). *If there is a function $f : \{0, 1\}^n \rightarrow \{0, 1\}$ in \mathcal{E} that is $1/3$ -hard for size $2^{\Omega(n)}$, then there is a function $f' : \{0, 1\}^m \rightarrow \{0, 1\}$ in \mathcal{E} that is $(1/2 - 2^{-\Omega(m)})$ -hard for size $2^{\Omega(m)}$.*

Proof. By Theorem 5.2 there exists a δ' -random function $g : \{0, 1\}^n \rightarrow \{0, 1\}$, where δ' is a constant, such that the hardness of $f' : \{0, 1\}^{O(n)} \rightarrow \{0, 1\}$ is $1/2 - \text{ExpBias}[(XOR \circ g^{\otimes k}) \circ IW_k] - 2^{-\Omega(n)}$ for circuits of size $2^{\Omega(n)}$.

We now bound the hardness. Whenever some $IW_i(x)$ falls in the set of inputs of density $2 \cdot \delta'$, where the output of g is a coin flip, the bias of $(XOR \circ g^{\otimes k}) \circ IW_k$ is 0. Therefore

$$\text{ExpBias}[(XOR \circ g^{\otimes k}) \circ IW_k] \leq \Pr_x[\forall i : IW_i(x) \notin H] \leq 2^{-\Omega(n)},$$

where in the last inequality we use standard hitting properties of expander walks (see, e.g., [11] for a proof) and take c to be a sufficiently large constant. \square

7. Limitations of monotone hardness amplification.

7.1. On the hypothesis that f is balanced. The hardness amplification results in the previous sections start from balanced functions. In this section we study this hypothesis. Our main finding is that, while this hypothesis is not necessary for hardness amplification within \mathcal{NP}/poly (i.e., nondeterministic polynomial size circuits), it is likely to be necessary for hardness amplification within \mathcal{NP} .

To see that this hypothesis is not necessary for amplification within \mathcal{NP}/poly , note that if the quantity $\Pr_x[f(x) = 1]$ of the original hard function $f : \{0, 1\}^n \rightarrow \{0, 1\}$ is known, then we can easily pad f to obtain a balanced function $\bar{f} : \{0, 1\}^{n+1} \rightarrow \{0, 1\}$:

$$\bar{f}(x, p) \stackrel{\text{def}}{=} \begin{cases} f(x) & \text{if } p = 0, \\ 0 & \text{if } p = 1 \text{ and } x \leq \Pr_x[f(x) = 1] \cdot 2^n, \\ 1 & \text{if } p = 1 \text{ and } x > \Pr_x[f(x) = 1] \cdot 2^n. \end{cases}$$

It is easy to see that \bar{f} is $1/\text{poly}(n)$ -hard if f is. Since a circuit can (nonuniformly) depend on $\Pr_x[f(x) = 1]$, the following hardness amplification within \mathcal{NP}/poly is a corollary to the proof of Theorem 4.1.

COROLLARY 7.1. *If there is a function $f : \{0, 1\}^n \rightarrow \{0, 1\}$ in \mathcal{NP}/poly that is $1/\text{poly}(n)$ -hard for size $s(n)$, then there is a function $f' : \{0, 1\}^m \rightarrow \{0, 1\}$ in \mathcal{NP}/poly that is $(1/2 - 1/s(\sqrt{m})^{\Omega(1)})$ -hard for size $s(\sqrt{m})^{\Omega(1)}$.*

Now we return to hardness amplification within \mathcal{NP} . First we note that, in our results, to amplify the hardness of $f : \{0, 1\}^n \rightarrow \{0, 1\}$ up to $1/2 - \epsilon$ it is necessary only that $\text{Bias}[f] \leq \epsilon^c$ for some universal constant c . The argument is standard and can be found, for example, in [34].

Combining this observation with the above padding technique, O'Donnell constructs several candidate hard functions, one for each “guess” of the bias of the original hard function. He then combines them in a single function using a different input length for each candidate; this gives a function that is very hard on average for infinitely many input lengths. However, this approach, even in conjunction with derandomization and nondeterminism, cannot give better hardness than $1/2 - 1/n$. (Roughly speaking, if we want to amplify to $1/2 - \epsilon$, then we will have at least $1/\epsilon$ different candidates, and thus the “hard” candidate may have input length $n \geq 1/\epsilon$, which means $1/2 - \epsilon \leq 1/2 - 1/n$.)

To what extent can we amplify the hardness of functions whose bias is *unknown*? Nonmonotone hardness amplifications, such as Yao’s XOR lemma, work regardless of the bias of the original hard function. However, in the rest of this section we show that, for hardness amplifications that are *monotone* and *black-box*, this is impossible. In particular, we show that black-box monotone hardness amplifications cannot amplify the hardness beyond the bias of the original function.

We now formalize the notion of black-box monotone hardness amplification and then state our negative result.

DEFINITION 7.2. *An oracle algorithm $\text{Amp} : \{0, 1\}^l \rightarrow \{0, 1\}$ is a black-box β -bias $[\delta \mapsto (1/2 - \epsilon)]$ -hardness amplification for length n and size s if for every $f : \{0, 1\}^n \rightarrow \{0, 1\}$ such that $\text{Bias}[f] \leq \beta$ and for every $A : \{0, 1\}^l \rightarrow \{0, 1\}$ such that*

$$\Pr[A(U_l) \neq \text{Amp}^f(U_l)] \leq \frac{1}{2} - \epsilon$$

there is an oracle circuit C of size at most s such that

$$\Pr[C^A(U_n) \neq f(U_n)] \leq \delta.$$

Amp is monotone if, for every x , $\text{Amp}^f(x)$ is a monotone function of the truth table of f .

Note that if Amp is as in Definition 7.2, and if f is δ -hard for size s' and $\text{Bias}[f] \leq \beta$, then Amp^f is $(1/2 - \epsilon)$ -hard for size s'/s : If there were a circuit A of size s'/s computing Amp^f with error probability at most $1/2 - \epsilon$, then C^A would be a circuit of size $s \cdot (s'/s) = s'$ computing f with error probability at most δ , contradicting the hardness of f . The term “black box” refers to the fact that the definition requires this to hold for *every* f and A , regardless of whether f is in \mathcal{NP} or A is a small circuit.

The following theorem shows that any *monotone* black-box hardness amplification up to $1/2 - \epsilon$ must start from functions of bias $\beta \approx \epsilon$.

THEOREM 7.3. *For any constant $\gamma > 0$, if Amp is a monotone black-box β -bias $[\delta \mapsto (1/2 - \epsilon)]$ -hardness amplification for length n and size $s \leq 2^{n/3}$ such that $1/2 - 4\epsilon > \delta + \gamma$, then $\beta \leq 8\epsilon + O(2^{-n})$.*

The main ideas for proving this bound are the same as in the negative result for black-box hardness amplification in [37]: First we show that the above kind of hardness amplification satisfies certain coding-like properties. Roughly, Amp can be seen as a kind of list-decodable code, where the distance property is guaranteed only for δ -distant messages with bias at most β (cf. [34]). Then we show that monotone functions fail to satisfy these properties. The limitation we prove on monotone functions relies on the following corollary to the Kruskal–Katona theorem (see [1, Theorem 7.3.1]).

LEMMA 7.4. *Let $\mathcal{S} = \{S_1, \dots, S_m\}$ be a collection of m subsets $S_i \subseteq \{1, \dots, N\}$, where $|S_i| = t$. If $m \geq \binom{N-1}{t} = (1 - \frac{t}{N}) \binom{N}{t}$, then for every integer $t' < t$*

$$|\{S : |S| = t', S \subseteq S_i \text{ for some } i\}| \geq \binom{N-1}{t'} = \left(1 - \frac{t'}{N}\right) \binom{N}{t'}.$$

Let \mathcal{F}_p be the uniform distribution on functions f whose truth tables have relative Hamming weight exactly p , i.e., $\Pr_x[f(x) = 1] = p$. We use the above Lemma 7.4 to prove the following fact.

LEMMA 7.5. *Let $A : \{0, 1\}^l \rightarrow \{0, 1\}$ be an oracle function such that, for every $x \in \{0, 1\}^l$, $A^f(x)$ is a monotone function of the truth table of $f : \{0, 1\}^n \rightarrow \{0, 1\}$. (For example, any monotone black-box hardness amplification Amp satisfies this condition.) Let τ be an integer multiple of $1/2^n$. Then there is $p \in \{1/2 - \tau, 1/2 + \tau\}$ such that*

$$\mathbb{E}_{U_l} [\text{Bias}_{F \leftarrow \mathcal{F}_p}[A^F(U_l)]] \geq \tau.$$

Proof. We show that for every fixed $x \in \{0, 1\}^l$ there is a $p \in \{1/2 - \tau, 1/2 + \tau\}$ such that $\text{Bias}_{F \leftarrow \mathcal{F}_p}[A^F(x)] \geq 2\tau$. The theorem then follows easily. Fix x . For every p define S_p as the set of functions f in \mathcal{F}_p such that $A^f(x) = 0$. Note that

$$\text{Bias}_{F \leftarrow \mathcal{F}_p}[A^F(x)] = \left|1 - \frac{2|S_p|}{|\mathcal{F}_p|}\right|.$$

If $|S_{1/2+\tau}| < (1/2 - \tau) \cdot |\mathcal{F}_{1/2+\tau}|$, then $\text{Bias}_{F \leftarrow \mathcal{F}_{1/2+\tau}}[A^F(x)] > 2\tau$ and we are done. Otherwise,

$$|S_{1/2+\tau}| \geq \left(\frac{1}{2} - \tau\right) \cdot |\mathcal{F}_{1/2+\tau}| = \left(1 - \left(\frac{1}{2} + \tau\right)\right) \cdot \binom{2^n}{(1/2 + \tau)2^n}.$$

View the elements $f \in S_p$ as subsets of $\{1, \dots, 2^n\}$ of size exactly $p2^n$ in the natural way; i.e., the subset associated with f is the set of inputs x such that $f(x) = 1$. Note that if $f' \subseteq f \subseteq \{1, \dots, 2^n\}$ and $A^f(x) = 0$, then by the monotonicity of A , $A^{f'}(x) = 0$. Therefore, by Lemma 7.4,

$$|S_{1/2-\tau}| \geq \left(1 - \left(\frac{1}{2} - \tau\right)\right) \cdot \binom{2^n}{(1/2 - \tau)2^n} = \left(\frac{1}{2} + \tau\right) \cdot |\mathcal{F}_{1/2-\tau}|,$$

and so $\text{Bias}_{F \leftarrow \mathcal{F}_{1/2-\tau}}[A^F(x)] \geq 2\tau$. \square

The following lemma captures the coding-like properties of monotone, black-box hardness amplifications—it shows that it is very unlikely that Amp^f for a “random f ” will land in any fixed Hamming ball of radius $1/2 - \epsilon$. For two functions f_1, f_2 , let

$Dist$ denote the relative Hamming distance of their truth tables, i.e., $Dist(f_1, f_2) \stackrel{\text{def}}{=} \Pr_x[f_1(x) \neq f_2(x)]$.

LEMMA 7.6. *Let $\gamma > 0$ be any fixed constant. Let Amp be a monotone black-box 8ϵ -bias $[\delta \mapsto (1/2 - \epsilon)]$ -hardness amplification for length n and size $s \leq 2^{n/3}$, where $1/2 - 4\epsilon > \delta + \gamma$, and let $\epsilon > 0$ be an integer multiple of $1/2^n$. Then for both p in $\{1/2 - 4\epsilon, 1/2 + 4\epsilon\}$ and every function G ,*

$$\Pr_{F \leftarrow \mathcal{F}_p} \left[Dist(G, Amp^F) \leq \frac{1}{2} - \epsilon \right] \leq 2^{-\Omega(2^n)}.$$

Proof. Let $N \stackrel{\text{def}}{=} 2^n$. For every function f of bias at most 8ϵ such that $Dist(G, Amp^f) \leq 1/2 - \epsilon$, there must exist a circuit of size s , with oracle access to G , that computes f with error at most δ . Therefore, since there are $2^{O(s \log s)}$ circuits of size s and no more than $2^{H(\delta)N}$ functions that are at distance at most δ from f , there are at most $2^{O(s \log s)} 2^{H(\delta)N}$ such functions. Thus, when we restrict our attention to the $\binom{N}{pN}$ functions in \mathcal{F}_p (for $p \in \{1/2 - 4\epsilon, 1/2 + 4\epsilon\}$), we have

$$\begin{aligned} \Pr_{F \leftarrow \mathcal{F}_p} \left[Dist(G, Amp^F) \leq \frac{1}{2} - \epsilon \right] &\leq \frac{2^{O(s \log s)} \cdot 2^{H(\delta)N}}{\binom{N}{pN}} \\ &\leq 2^{O(s \log s)} \cdot (N + 1) \cdot 2^{(H(\delta) - H(p))N} \\ &\leq 2^{O(s \log s)} \cdot (N + 1) \cdot 2^{(H(\delta) - H(1/2 - 4\epsilon))N} \\ &\leq 2^{-\Omega(N)}, \end{aligned}$$

where the second inequality follows from the fact that $\binom{N}{pN} \geq 2^{H(p)N} / (N + 1)^8$ and the last inequality uses the fact that $1/2 - 4\epsilon > \delta + \gamma$ implies $H(1/2 - 4\epsilon) > H(\delta) + \Omega(1)$. \square

Proof of Theorem 7.3. We assume that ϵ is a positive integer multiple of $1/2^n$ and show that $\beta \leq 8\epsilon$. The theorem then follows for general ϵ by rounding it up to the next integer multiple of $1/2^n$. We show that $\beta = 8\epsilon$ is impossible, and therefore that $\beta < 8\epsilon$ (because any β' -bias hardness amplification is clearly also a β -bias hardness amplification for every $\beta \leq \beta'$).

Suppose, for the sake of contradiction, that $\beta = 8\epsilon$.

By Lemma 7.5, we may choose $p \in \{1/2 - 4\epsilon, 1/2 + 4\epsilon\}$ such that

$$\mathbb{E}_{U_l} [\text{Bias}_{F \leftarrow \mathcal{F}_p}[Amp^F(U_l)]] \geq 4\epsilon.$$

Define the function $G(x) \stackrel{\text{def}}{=} \text{Maj}_{F \leftarrow \mathcal{F}_p} Amp^F(x)$, and consider

$$(7) \quad \Pr_{U_l, F \leftarrow \mathcal{F}_p} [Amp^F(U_l) \neq G(U_l)].$$

⁸Actually, $\binom{N}{pN}$ is asymptotically $\Theta(2^{H(p)N} / \sqrt{N})$, but for our purposes the easier estimate stated suffices.

We have

$$\begin{aligned}
 & \Pr_{U_l, F \leftarrow \mathcal{F}_p} [Amp^F(U_l) \neq G(U_l)] \\
 &= \mathbb{E}_{U_l} \left[\Pr_{F \leftarrow \mathcal{F}_p} [Amp^F(U_l) \neq G(U_l)] \right] \\
 &= \mathbb{E}_{U_l} \left[\frac{1}{2} - \frac{\text{Bias}_{F \leftarrow \mathcal{F}_p}[Amp^F(U_l)]}{2} \right] \quad (\text{by def. of bias and } G) \\
 &= \frac{1}{2} - \frac{\mathbb{E}_{U_l} [\text{Bias}_{F \leftarrow \mathcal{F}_p}[Amp^F(U_l)]]}{2} \\
 &\leq \frac{1}{2} - 2\epsilon. \quad (\text{by the choice of } p)
 \end{aligned}$$

On the other hand, Lemma 7.6 implies that quantity (7) is at least $1/2 - \epsilon - 2^{-\Omega(2^n)}$ (note that the hypothesis of the lemma is satisfied by our assumption that $1/2 - 4\epsilon \geq \delta + \gamma$).

Combining the two bounds, we have that

$$\frac{1}{2} - 2\epsilon \geq \Pr_{U_l, F \leftarrow \mathcal{F}_p} [Amp^F(U_l) \neq G(U_l)] \geq \frac{1}{2} - \epsilon - 2^{-\Omega(2^n)},$$

which is a contradiction for sufficiently large n (by the assumption that ϵ is a positive multiple of $1/2^n$). \square

7.2. Nondeterminism is necessary. In this subsection we show that *deterministic*, monotone, nonadaptive, black-box hardness amplifications cannot amplify hardness beyond $1/2 - 1/\text{poly}(n)$. Thus, the use of *nondeterminism* in our results (section 5.4) seems necessary. Note that most hardness amplifications, including the one in this paper, are black-box and nonadaptive.

O’Donnell [31] proves that any monotone “direct product construction” (i.e., $f'(x_1, \dots, x_k) = C(f(x_1), \dots, f(x_k))$, as in (1)) cannot amplify to hardness better than $1/2 - 1/n$, assuming that the amplification works for all functions f (not necessarily in \mathcal{NP}). We relax the assumption that the hardness amplification is a direct product construction (allowing any monotone nonadaptive oracle algorithm $f' = Amp^f$). On the other hand, we require that the reduction proving its correctness also be black-box (as formalized in Definition 7.2).

We prove our bound even for hardness amplifications that amplify only balanced functions (i.e., $\beta = 0$ in Definition 7.2).

THEOREM 7.7. *For every constant $\delta < 1/2$, if Amp is a black-box 0-bias $[\delta \mapsto (1/2 - \epsilon)]$ -hardness amplification for length n and size $s \leq 2^{n/3}$ such that, for every x , $Amp^f(x)$ is a monotone function of $k \leq 2^{n/3}$ values of f , then*

$$\epsilon \geq \Omega\left(\frac{\log^2 k}{k}\right).$$

The proof of this result follows closely the proof of the negative result on hardness amplification in [37]. The main difference is that here we use bounds on the noise stability of monotone functions rather than constant depth circuits.

By a Chernoff bound, a $2^{-\epsilon^2 N}$ fraction of 2^N strings are ϵ -unbalanced. But it is impossible to kill this fraction with noise bounded away from $1/2$. (This will only give min entropy $2^{-\epsilon N}$, which will not kill $2^{-\epsilon^2 N + N}$.)

The following lemma is similar to Lemma 7.6. The only difference is in considering functions F at distance η from f ; this will correspond to perturbing the monotone amplification function with noise having parameter η .

LEMMA 7.8. *Let Amp be as in Definition 7.2 with $\beta = 0$ and $s \leq 2^{n/3}$. Then for any constant $\delta < 1/2$ there is a constant $\eta < 1/2$ such that, for sufficiently large n , the following holds: If $f : \{0, 1\}^n \rightarrow \{0, 1\}$ is any fixed balanced function and $F : \{0, 1\}^n \rightarrow \{0, 1\}$ is a random balanced function such that $\text{Dist}(f, F) = \eta$, then*

$$\Pr_F \left[\text{Dist}(\text{Amp}^f, \text{Amp}^F) \leq \frac{1}{2} - \epsilon \right] \leq \epsilon.$$

Proof. Let $N \stackrel{\text{def}}{=} 2^n$. It is easy to see that F is uniform on a set of size $\binom{N/2}{\eta N/2}^2$. The rest of the proof is like the proof of Lemma 7.6:

$$\begin{aligned} \Pr \left[\text{Dist}(\text{Amp}^f, \text{Amp}^F) \leq \frac{1}{2} - \epsilon \right] &\leq \frac{2^{O(s \log s)} 2^{H(\delta)N}}{\binom{N/2}{\eta N/2}^2} \\ &\leq 2^{O(s \log s)} \cdot \left(\frac{N}{2} + 1 \right)^2 \cdot 2^{(H(\delta) - H(\eta))N} \\ &\leq \epsilon, \end{aligned}$$

where the last inequality holds for a suitable choice of $\eta < 1/2$, using the fact that $\delta < 1/2$ is a constant and that $s \leq 2^{n/3}$. \square

Proof of Theorem 7.7. Let η be the constant in Lemma 7.8. The idea is to consider

$$(8) \quad \Pr_{U_l, F, F'} [\text{Amp}^F(U_l) \neq \text{Amp}^{F'}(U_l)],$$

where F is a random balanced function and F' is a random balanced function such that $\text{Dist}(F, F') = \eta$.

By the above lemma, the probability (8) is at least $1/2 - 2\epsilon$.

On the other hand, for every fixed x , $\text{Amp}^F(x)$ is a monotone function depending only on k bits of the truth table of the function F . Since k is small compared to 2^n , the distribution (F, F') induces on the input of $\text{Amp}^F(x)$ a distribution very close to $(U_k, U_k \oplus \mu)$, where μ is a noise vector with parameter η . Specifically, it can be verified that the statistical difference between these two distributions is at most $O(k^2/(\eta 2^n))$. Because this value is dominated by $\log^2 k/k$ when $k \leq 2^{n/3}$, and because $\text{Amp}^F(x)$ is a *monotone* function of k bits, we may apply Theorem 3.10 to conclude that the probability (8) is at most $1/2 - O(\log^2 k/k)$.

Combining the two bounds, we have that $1/2 - O(\log^2 k/k) \geq 1/2 - 2\epsilon$, and the results follows. \square

Acknowledgments. We thank Ryan O’Donnell and Rocco Servedio for an email exchange about noise stability. We thank Richard Stanley for pointing out the Kruskal–Katona theorem. We also thank Oded Goldreich, Luca Trevisan, Avi Wigderson, and the anonymous reviewers for helpful suggestions.

REFERENCES

[1] I. ANDERSON, *Combinatorics of Finite Sets*, 2nd ed., Dover Publications, Mineola, NY, 2002.
 [2] L. BABAI, L. FORTNOW, AND C. LUND, *Nondeterministic exponential time has two-prover interactive protocols*, *Comput. Complexity*, 1 (1991), pp. 3–40.

- [3] L. BABAI, L. FORTNOW, N. NISAN, AND A. WIGDERSON, *BPP has subexponential time simulations unless EXPTIME has publishable proofs*, *Comput. Complexity*, 3 (1993), pp. 307–318.
- [4] D. BEAVER AND J. FEIGENBAUM, *Hiding instances in multioracle queries*, in *Proceedings of the 7th Annual Symposium on Theoretical Aspects of Computer Science*, *Lecture Notes in Comput. Sci.* 415, Springer, New York, 1990, pp. 37–48.
- [5] M. BEN-OR AND N. LINIAL, *Collective coin-flipping*, in *Randomness and Computation*, Silvio Micali, ed., Academic Press, New York, 1990, pp. 91–115.
- [6] A. BOGDANOV AND L. TREVISAN, *On worst-case to average-case reductions for NP problems*, in *Proceedings of the 44th Annual Symposium on Foundations of Computer Science*, Cambridge, MA, 2003, IEEE Press, Piscataway, NJ, 2003, pp. 308–317.
- [7] J.-Y. CAI, A. PAVAN, AND D. SIVAKUMAR, *On the hardness of the permanent*, in *Proceedings of the 16th International Symposium on Theoretical Aspects of Computer Science*, *Lecture Notes in Comput. Sci.* 1563, Springer-Verlag, New York, 1999, pp. 90–99.
- [8] U. FEIGE AND C. LUND, *On the hardness of computing the permanent of random matrices*, *Comput. Complexity*, 6 (1996), pp. 101–132.
- [9] J. FEIGENBAUM AND L. FORTNOW, *Random-self-reducibility of complete sets*, *SIAM J. Comput.*, 22 (1993), pp. 994–1005.
- [10] L. FORTNOW, *Balanced NP sets*, *Computational Complexity Weblog*, 2003, available online at http://weblog.fortnow.com/archive/2003_09_07_archive.html.
- [11] O. GOLDREICH, *A sample of samplers—A computational perspective on sampling (survey)*, Technical report TR97-020, Electronic Colloquium on Computational Complexity (ECCC), 1997.
- [12] O. GOLDREICH, *Modern Cryptography, Probabilistic Proofs, and Pseudorandomness*, *Algorithms Combin.* 17, Springer-Verlag, Berlin, 1999.
- [13] O. GOLDREICH, *Foundations of Cryptography: Volume 1, Basic Tools*, Cambridge University Press, Cambridge, UK, 2001.
- [14] O. GOLDREICH AND L. A. LEVIN, *A hard-core predicate for all one-way functions*, in *Proceedings of the 21st Annual ACM Symposium on Theory of Computing*, Seattle, WA, 1989, ACM, New York, pp. 25–32.
- [15] O. GOLDREICH, N. NISAN, AND A. WIGDERSON, *On Yao’s XOR Lemma*, Technical Report TR95-050, Electronic Colloquium on Computational Complexity, 1995, available online at <http://www.eccc.uni-trier.de/eccc>.
- [16] D. GUTFREUND AND E. VIOLA, *Fooling parity tests with parity gates*, in *Proceedings of the Eighth International Workshop on Randomization and Computation (RANDOM)*, *Lecture Notes in Comput. Sci.* 3122, Springer-Verlag, New York, 2004, pp. 381–392.
- [17] A. HEALY, S. VADHAN, AND E. VIOLA, *Using nondeterminism to amplify hardness*, in *Proceedings of the 36th Annual ACM Symposium on the Theory of Computing*, Chicago, IL, 2004, ACM, New York, pp. 192–201.
- [18] R. IMPAGLIAZZO, *Hard-core distributions for somewhat hard problems*, in *Proceedings of the 36th Annual Symposium on Foundations of Computer Science*, IEEE, 1995, pp. 538–545.
- [19] R. IMPAGLIAZZO, *A personal view of average-case complexity*, in *Proceedings of the 10th Annual Structure in Complexity Theory Conference*, Milwaukee, WI, 1995, IEEE Press, Piscataway, NJ, 1995, pp. 134–147.
- [20] R. IMPAGLIAZZO AND A. WIGDERSON, *$P = BPP$ if E requires exponential circuits: Derandomizing the XOR lemma*, in *Proceedings of the 29th Annual ACM Symposium on Theory of Computing*, El Paso, TX, 1997, ACM, New York, pp. 220–229.
- [21] R. IMPAGLIAZZO AND A. WIGDERSON, *Randomness vs. time: Derandomization under a uniform assumption*, *J. Comput. System Sci.*, 63 (2001), pp. 672–688.
- [22] J. KAHN, G. KALAI, AND N. LINIAL, *The influence of variables on Boolean functions (extended abstract)*, in *Proceedings of the 29th Annual Symposium on Foundations of Computer Science*, White Plains, New York, IEEE Press, Piscataway, NJ, 1988, pp. 68–80.
- [23] A. KLIVANS AND R. A. SERVEDIO, *Boosting and hard-core sets*, *Machine Learning*, 53 (2003), pp. 217–238.
- [24] L. A. LEVIN, *Average case complete problems*, *SIAM J. Comput.*, 15 (1986), pp. 285–286.
- [25] R. LIPTON, *New directions in testing*, in *Proceedings of the DIMACS Workshop on Distributed Computing and Cryptography*, vol. 2, ACM/AMS, 1991, pp. 191–202.
- [26] C.-J. LU, S.-C. TSAI, AND H.-L. WU, *On the complexity of hardness amplification*, in *Proceedings of the 20th Annual IEEE Conference on Computational Complexity*, San Jose, CA, IEEE Press, Piscataway, NJ, 2005, pp. 170–182.
- [27] E. MOSSEL AND R. O’DONNELL, *On the noise sensitivity of monotone functions*, *Random Struct. Algorithms*, 23 (2003), pp. 333–350.
- [28] N. NISAN, *Pseudorandom bits for constant depth circuits*, *Combinatorica*, 11 (1991), pp. 63–70.

- [29] N. NISAN, *Pseudorandom generators for space-bounded computation*, *Combinatorica*, 12 (1992), pp. 449–461.
- [30] N. NISAN AND A. WIGDERSON, *Hardness vs. randomness*, *J. Comput. System Sci.*, 49 (1994), pp. 149–167.
- [31] R. O'DONNELL, *Hardness amplification within NP*, *J. Comput. System Sci.*, 69 (2004), pp. 68–94.
- [32] R. SHALTIEL AND C. UMANS, *Simple extractors for all min-entropies and a new pseudorandom generator*, *J. ACM*, 52 (2005), pp. 172–216.
- [33] M. SUDAN, L. TREVISAN, AND S. VADHAN, *Pseudorandom generators without the XOR lemma*, *J. Comput. System Sci.*, 62 (2001), pp. 236–266.
- [34] L. TREVISAN, *List decoding using the XOR lemma*, in *Proceedings of the 44th Annual Symposium on Foundations of Computer Science*, Cambridge, MA, 2003, IEEE Press, Piscataway, NJ, 2003, pp. 126–135.
- [35] L. TREVISAN AND S. VADHAN, *Pseudorandomness and average-case complexity via uniform reductions*, in *Proceedings of the 17th Annual IEEE Conference on Computational Complexity*, Montreal, QC, IEEE Press, Piscataway, NJ, 2002, pp. 129–138.
- [36] C. UMANS, *Pseudo-random generators for all hardnesses*, *J. Comput. System Sci.*, 67 (2003), pp. 419–440.
- [37] E. VIOLA, *The complexity of constructing pseudorandom generators from hard functions*, *Comput. Complexity*, 13 (2004), pp. 147–188.
- [38] E. VIOLA, *On constructing parallel pseudorandom generators from one-way functions*, in *Proceedings of the 20th Annual IEEE Conference on Computational Complexity*, San Jose, CA, IEEE Press, Piscataway, NJ, 2005, pp. 183–197.
- [39] A. C. YAO, *Theory and applications of trapdoor functions (extended abstract)*, in *Proceedings of the 23rd Annual Symposium on Foundations of Computer Science*, Chicago, IL, IEEE, 1982, pp. 80–91.

LOGARITHMIC LOWER BOUNDS IN THE CELL-PROBE MODEL*

MIHAI PĂTRAȘCU[†] AND ERIK D. DEMAINÉ[†]

Abstract. We develop a new technique for proving cell-probe lower bounds on dynamic data structures. This technique enables us to prove an amortized randomized $\Omega(\lg n)$ lower bound per operation for several data structural problems on n elements, including partial sums, dynamic connectivity among disjoint paths (or a forest or a graph), and several other dynamic graph problems (by simple reductions). Such a lower bound breaks a long-standing barrier of $\Omega(\lg n / \lg \lg n)$ for any dynamic language membership problem. It also establishes the optimality of several existing data structures, such as Sleator and Tarjan’s dynamic trees. We also prove the first $\Omega(\log_B n)$ lower bound in the external-memory model without assumptions on the data structure (such as the comparison model). Our lower bounds also give a query-update trade-off curve matched, e.g., by several data structures for dynamic connectivity in graphs. We also prove matching upper and lower bounds for partial sums when parameterized by the word size and the maximum additive change in an update.

Key words. cell-probe complexity, lower bounds, data structures, dynamic graph problems, partial-sums problem

AMS subject classification. 68Q17

DOI. 10.1137/S0097539705447256

1. Introduction. The cell-probe model is perhaps the strongest model of computation for data structures, subsuming in particular the common word-RAM model. We suppose that the memory is divided into fixed-size cells (words), and the cost of an operation is just the number of cells it reads or writes. Typically we think of the cell size as being around $\lg n$ bits long, so that a single cell can address all n elements in the data structure. (Refer to section 4 for a precise definition of the model.) While the cell-probe model is unrealistic as a model of computation for actual data structures, its generality makes it an important model for lower bounds on data structures.

Previous cell-probe lower bounds for data structures fall into two categories of approaches. The first approach is based on communication complexity. Lower bounds for the predecessor problem [Ajt88, MNSW98, BF02, SV] are perhaps the most successful application of this idea. Unfortunately, this approach can be applied only to problems that are hard even in the static case. It also requires queries to receive a parameter of $\omega(\lg n)$ bits, which is usually interpreted as requiring cells to have $\omega(\lg n)$ bits. For problems that are hard only in the dynamic case, all lower bounds have used some variation of the chronogram method of Fredman and Saks [FS89]. By design, this method cannot prove a trade-off between the query time t_q and the update time t_u better than $t_q \lg t_u = \Omega(\lg n)$, which was achieved for the marked-ancestor problem (and consequently many other problems) in [AHR98]. This limitation on trade-off lower bounds translates into an $\Omega(\lg n / \lg \lg n)$ limitation on lower bounds for both queries and updates provable by this technique. The $\Omega(\lg n / \lg \lg n)$ barrier

*Received by the editors October 31, 2004; accepted for publication (in revised form) May 29, 2005; published electronically February 21, 2006. This work is based on the following two conference publications by the same authors: *Lower bounds for dynamic connectivity*, in Proceedings of the 36th Annual ACM Symposium on Theory of Computing (STOC '04), pp. 546–553, and *Tight bounds for the partial-sums problem*, in Proceedings of the 15th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA '04), pp. 20–29.

<http://www.siam.org/journals/sicomp/35-4/44725.html>

[†]MIT Computer Science and Artificial Intelligence Laboratory, 32 Vassar St., Cambridge, MA 02139 (mip@mit.edu, edemaine@mit.edu).

has been recognized as an important limitation in the study of data structures and was proposed as a major challenge for future research in a recent survey [Mil99].

This paper introduces a new technique for proving cell-probe lower bounds on dynamic data structures. With this technique we establish an $\Omega(\lg n)$ lower bound for either queries or updates in several natural and well-studied problems, in particular, maintaining partial (prefix) sums in an array and dynamic connectivity among disjoint paths (or a forest or a graph). (We detail the exact problems we consider, and all results we obtain, in section 2; we summarize relevant previous results on these problems in section 3.) These lower bounds establish the optimality of several data structures, including the folkloric $O(\lg n)$ balanced tree data structure for partial sums, and Sleator and Tarjan's dynamic trees data structure (which in particular maintains dynamic connectivity in a forest).

We also prove a trade-off lower bound of $t_q \lg \frac{t_u}{t_q} = \Omega(\lg n)$.¹ This trade-off turns out to be the right answer for our problems, and implies the $\Omega(\lg n)$ bound on the worst of queries and updates. In addition, we can prove a symmetric trade-off $t_u \lg \frac{t_q}{t_u} = \Omega(\lg n)$. As mentioned above, it is fundamentally impossible to achieve such a trade-off using the previous techniques.

We also refine our analysis of the partial-sums problem beyond the dependence on n . Specifically, we parameterize by n , the number b of bits in a word, and the number δ of bits in an update. Naturally, $\delta \leq b$, but in some applications, δ is much smaller than b . We prove tight upper and lower bounds of $\Theta(\frac{\lg n}{\lg(b/\delta)})$ on the worst of queries and updates. This result requires improvements in both the upper bounds and the lower bounds. In addition, we give the tight query/update trade-off $t_q(\lg \frac{b}{\delta} + \lg \frac{t_u}{t_q}) = \Theta(\lg n)$. The tightness of this characterization is particularly unusual given its dependence on five variables.

The main idea behind our lower bound technique is to organize time (the sequence of operations performed on the data structure) into a complete tree. The heart of the analysis is an encoding/decoding argument that bounds the amount of information transferred between disjoint subtrees of the tree: if few cells are read and written, then little information can be transferred. The nature of the problems of interest requires at least a certain amount of information transfer from updates to queries, providing a lower bound on the number of cells read and written. This main idea is developed first in section 5 in the context of the partial-sums problem, where we obtain a short proof of an $\Omega(\lg n)$ lower bound for partial sums. Compared to the lower bounds based on previous techniques, our technique leads to relatively clean proofs with minimal combinatorial calculation.

We generalize this basic approach in several directions to obtain our further lower bounds. In section 6, we show how our technique can be extended to handle queries with binary answers (such as dynamic connectivity) instead of word-length answers (such as partial sums). In particular, we obtain an $\Omega(\lg n)$ lower bound for dynamic connectivity in disjoint paths. We also show how to use our lower bound technique in the presence of nondeterminism or Monte Carlo randomization. In section 7, we show how our technique can be further extended to handle updates asymptotically smaller than the word size, in particular obtaining lower bounds for the partial-sums problem when $\delta < b$ and for dynamic connectivity in the external-memory model. This section develops the most complicated form of our technique.

The final sections contain complementary results to this main flow of the lower

¹Throughout this paper, $\lg x$ denotes $\log_2(2+x)$, which is positive for all $x \geq 0$ (an important property in this bound and several others).

bound technique. In section 8, we give tight upper bounds for the partial-sums problem. The data structure is based on a few interesting ideas that enable us to eliminate the precomputed tables from previous approaches. In section 9, we prove some easy reductions from dynamic connectivity to other dynamic graph problems, transferring our lower bounds to these problems. Finally, we conclude in section 10 with a list of open problems.

2. Results. In this section we give precise descriptions of the problems we consider, our results, and a brief synopsis of how these results compare to previous work. (Section 3 gives a more detailed historical account.)

2.1. The partial-sums problem. This problem asks to maintain an array $A[1..n]$ of n integers subject to the following operations:

UPDATE(k, Δ): modify $A[k] \leftarrow \Delta$.

SUM(k): returns the partial sum $\sum_{i=1}^k A[i]$.

SELECT(σ): returns an index i satisfying $\text{SUM}(i-1) < \sigma \leq \text{SUM}(i)$. To guarantee uniqueness of the answer, we require that $A[i] > 0$ for all i .

Besides n , the problem has several interesting parameters. One parameter is b , the number of bits in a cell (word). We assume that every array element and sum fits in a cell. Also, we assume that $b = \Omega(\lg n)$. Another parameter is δ , the number of bits needed to represent an argument Δ to **UPDATE**. Naturally, δ is bounded above by b ; however, it is traditional (see, e.g., [RRR01]) to consider a separate parameter δ because it is smaller in many applications. We write t_u for the running time of **UPDATE**, t_q for the running time of **SUM**, and t_s for the running time of **SELECT**.

We first study the unrestricted case when $\delta = \Omega(b)$.

THEOREM 2.1. *Consider any cell-probe data structure for the partial-sums problem that may use amortization and Las Vegas randomization. If $\delta = \Omega(b)$, then the following trade-offs hold:*

$$\begin{aligned} t_q \lg(t_u/t_q) &= \Omega(\lg n); & t_u \lg(t_q/t_u) &= \Omega(\lg n); \\ t_s \lg(t_u/t_s) &= \Omega(\lg n); & t_u \lg(t_s/t_u) &= \Omega(\lg n). \end{aligned}$$

The trade-off curves are identical for the **SELECT** and **SUM** operations. The first branch of each trade-off is relevant when queries are faster than updates, while the second branch is relevant when updates are faster. The trade-offs imply the long-sought logarithmic bound for the partial-sums problem: $\max\{t_u, t_q\} = \Omega(\lg n)$. The best previous bound, by Fredman and Saks [FS89], was $t_q \lg(bt_u) = \Omega(\lg n)$, implying $\max\{t_u, t_q\} = \Omega(\lg n / \lg b)$. The trade-off curves between t_u and t_q also hold in the group model of computation, where elements of the array come from a black-box group and time is measured as the number of algebraic operations. The best previous bound for this model was $\Omega(\lg n / \lg \lg n)$, also by Fredman and Saks [FS89].

A classic result achieves $t_u = t_q = t_s = O(\lg n)$. For the **SUM** query, our entire trade-off curve can be matched (again, this is folklore; see the next section on previous work). For **SELECT**, the trade-offs cannot be tight for the entire range of parameters because, even for polynomial update times, there is a superconstant lower bound on the query time for the predecessor problem [Ajt88, SV].

We also analyze the case $\delta = o(b)$. We first give the following lower bounds.

THEOREM 2.2. *Consider any cell-probe data structure for the partial-sums problem using amortization and Las Vegas randomization. The following trade-offs hold: $t_q(\lg \frac{b}{\delta} + \lg \frac{t_u}{t_q}) = \Omega(\lg n)$ and $t_s(\lg \frac{b}{\delta} + \lg \frac{t_u}{t_s}) = \Omega(\lg n)$.*

In this case, we cannot prove a reverse trade-off (when updates are faster than queries). This trade-off implies the rather interesting lower bound $\max\{t_u, t_q\} = \Omega(\frac{\lg n}{\lg(b/\delta)})$, and similarly for t_s . When $\delta = \Theta(b)$, this gives the same $\Omega(\lg n)$ as before. We give new matching upper bounds as follows.

THEOREM 2.3. *There exists a data structure for the partial-sums problem achieving $t_u = t_q = t_s = O(\frac{\lg n}{\lg(b/\delta)})$. The data structure runs on a RAM, is deterministic, and achieves worst-case bounds.*

Our upper bounds can handle a slightly harder version of the problem, where **UPDATE**(i, Δ) has the effect $A[i] \leftarrow A[i] + \Delta$. Thus, we are not restricting each $A[i]$ to δ bits, but just mandate that they don't grow by more than a δ -bit term at a time. Several previous results [Die89, RRR01, HSS03] achieved $O(\lg n / \lg \lg n)$ bounds for $\delta = O(\lg \lg n)$. None of these solutions scale well with δ or b because they require large precomputed tables.

This result not only matches the previous lower bound on the hardest operation but also actually helps match the entire trade-off of Theorem 2.2. Indeed, the trade-off lower bound shows that there is effectively no interesting trade-off when $\delta = o(b)$: when $\frac{b}{\delta} \geq \frac{t_u}{t_q}$, the tight bound is $t_q = \Theta(\frac{\lg n}{\lg(b/\delta)})$, matched by our structure; when $\frac{b}{\delta} < \frac{t_u}{t_q}$, the tight bound is $t_q = \Theta(\frac{\lg n}{\lg(t_u/t_q)})$, matched by the classic result, which does not depend on δ . Thus, we obtain an unusually precise understanding of the problem, having a trade-off that is tight in all five parameters.

2.2. Dynamic connectivity. This problem asks to maintain an undirected graph with a fixed set of n vertices subject to the following operations:

INSERT(u, v): insert an edge (u, v) into the graph.

DELETE(u, v): delete the edge (u, v) from the graph.

CONNECTED(u, v): test whether u and v lie in the same connected component.

We write t_q for the running time of **CONNECTED** and t_u for the running times of **INSERT** and **DELETE**. It makes the most sense to study this problem in the cell-probe model with $O(\lg n)$ bits per cell because every quantity in the problem occupies $O(\lg n)$ bits.

We prove the following lower bound.

THEOREM 2.4. *Any cell-probe data structure for dynamic connectivity satisfies the following trade-offs: $t_q \lg(t_u/t_q) = \Omega(\lg n)$ and $t_u \lg(t_q/t_u) = \Omega(\lg n)$. These bounds hold under amortization, nondeterministic queries, and Las Vegas randomization, or under Monte Carlo randomization with error probability $n^{-\Omega(1)}$. These bounds hold even if the graph is always a disjoint union of paths.*

This lower bound holds under very broad assumptions. It allows for nondeterministic computation or Monte Carlo randomization (with polynomially small error) and holds even for paths (and thus for trees, plane graphs, etc.). The trade-offs we obtain are identical to the partial-sums problem. In particular, we obtain that $\max\{t_u, t_q\} = \Omega(\lg n)$.

An upper bound of $O(\lg n)$ for trees is given by the famous dynamic trees data structure of Sleator and Tarjan [ST83]. In addition, the entire trade-off curve for $t_u = \Omega(t_q)$ can be matched for trees. For general graphs, Thorup [Tho00] gave an almost-matching upper bound of $O(\lg n (\lg \lg n)^3)$. For any $t_u = \Omega(\lg n (\lg \lg n)^3)$, his data structure can match our trade-off.

Dynamic connectivity is perhaps the most fundamental dynamic graph problem. It is relatively easy to show by reductions that our bounds hold for several other dynamic graph problems. Section 9 describes such reductions for deciding connectivity of the entire graph, minimum spanning forest, and planarity testing. Many

data structural problems on undirected graphs have polylogarithmic solutions, so our bound is arguably interesting for these problems. Some problems have logarithmic solutions for special cases (such as plane graphs), and our results prove optimality of those data structures.

We also consider dynamic connectivity in the external-memory model. Let B be the page (block) size, i.e., the number of $(\lg n)$ -bit cells that fit on one page. We prove the following lower bound.

THEOREM 2.5. *A data structure for dynamic connectivity in the external-memory model with page size B must satisfy $t_q(\lg B + \lg \frac{t_u}{t_q}) = \Omega(\lg n)$. This bound allows for amortization, Las Vegas randomization, and nondeterminism, and holds even if the graph is always a disjoint union of paths.*

Thus we obtain a bound of $\max\{t_u, t_q\} = \Omega(\log_B n)$. Although bounds of this magnitude are ubiquitous in the external-memory model, our lower bound is the first that holds in a general model of computation, i.e., allowing data items to be manipulated arbitrarily and just counting the number of page transfers. Previous lower bounds have assumed the comparison model or indivisibility of data items.

It is possible to achieve an $O(\log_B n)$ upper bound for a forest by combining Euler tour trees with buffer trees [Arg03]. As with the partial-sums problem, this result implies that our entire trade-off is tight for trees: for $B \geq t_u/t_q$, this solution is optimal; if the term in t_u/t_q dominates, we use the classic trade-off, which foregoes the benefit of memory pages.

3. Previous work. In this section we detail the relevant history of cell-probe lower bounds in general and the specific problems we consider.

3.1. Cell-probe lower bounds. Fredman and Saks [FS89] were the first to prove cell-probe lower bounds for dynamic data structures. They developed the chronogram technique and used it to prove a lower bound of $\Omega(\lg n / \lg b)$ for the partial-sums problem in $\mathbb{Z}/2\mathbb{Z}$ (integers modulo 2, where elements are bits and addition is equivalent to binary exclusive-or). This bound assumes $b \geq \lg n$ so that an index into the n -element array fits in a word; for the typical case of $b = \Theta(\lg n)$, it implies an $\Omega(\lg n / \lg \lg n)$ lower bound. Fredman and Saks also obtain a trade-off of $t_q = \Omega(\frac{\lg n}{\lg b + \lg t_u})$.

There has been considerable exploration of what the chronogram technique can offer. Ben-Amram and Galil [BAG01] reprove the lower bounds of Fredman and Saks in a more formalized framework built around the concepts of problem and output variability. Using these ideas, they show in [BAG02] that the lower bound holds even if cells have infinite precision, but the set of operations is restricted.

Miltersen et al. [MSVT94] observe that there is a trivial reduction from the partial-sums problem in $\mathbb{Z}/2\mathbb{Z}$ to dynamic connectivity, implying an $\Omega(\lg n / \lg \lg n)$ lower bound for the latter problem. Independently, Fredman and Henzinger [FH98] observe the same reduction, as well as more complex reductions applied to connectivity in plane graphs and dynamic planarity testing. Husfeldt and Rauhe [HR03] show slightly stronger results using the chronogram technique. They prove that the lower bound holds even for nondeterministic algorithms, and even in a promise version of the problem in which the algorithm is told the requested sum to ± 1 precision. These improved results make it possible to prove reductions to various other problems [HR03, HRS96].

Alstrup, Husfeldt, and Rauhe [AHR98] give the only previous improvement to the bounds of Fredman and Saks by proving a stronger trade-off of $t_q \lg t_u = \Omega(\lg n)$. This

bound is the best trade-off provable by the chronogram technique. However, it still cannot improve beyond $\max\{t_u, t_q\} = \Omega(\lg n / \lg \lg n)$. The problem they considered was the partial-sums problem generalized to trees, where a query asks for the sum of a root-to-leaf path. (This is a variation of the more commonly known marked-ancestor problem.) Their bound is tight for balanced trees; for arbitrary trees, our lower bound shows that $\Theta(\lg n)$ is the best possible.

Miltersen [Mil99] surveys the field of cell-probe complexity and advocates “dynamic language membership” problems as a standardized framework for comparing lower bounds. Given a language L that is polynomial-time decidable, the *dynamic language membership problem* for L is defined as follows. For any given n (the problem size), maintain a string $w \in \{0, 1\}^n$ under two operations: flip the i th bit of w , and report whether $w \in L$. Through its minimalism, this framework avoids several pitfalls in comparing lower bounds. For instance, it is possible to prove very high lower bounds in terms of the number of cells in the problem representation (which, misleadingly, is often denoted n), if the cells are large [Mil99]. However, these lower bounds are not very interesting because they assume exponential-size cells. In terms of the number of bits in the problem representation, all known lower bounds do not exceed $\Omega(\lg n / \lg \lg n)$.

Miltersen proposes several challenges for future research, two of which we solve in this paper. One such challenge was to prove an $\Omega(\lg n)$ lower bound for the partial-sums problem. Another such challenge, listed as one of three “big challenges,” was to prove a lower bound of $\omega(\lg n / \lg \lg n)$ for a dynamic language membership problem. We solve this problem because dynamic connectivity can be phrased as a dynamic language membership problem [Mil99].

3.2. The partial-sums problem in other models. The partial-sums problem has been studied since the dawn of data structures and has served as the prototypical problem for the study of lower bounds. Initial efforts concentrated on algebraic models of computation. In the semigroup or group models, the elements of the array come from a black-box (semi)group. The algorithm can manipulate the Δ inputs only through additions and, in the group model, subtractions; all other computations in terms of the indices touched by the operations are free.

In the semigroup model, Fredman [Fre81] gives a tight logarithmic bound. However, this bound is generally considered weak, because updates have the form $A[i] \leftarrow \Delta$. Because additive inverses do not exist, such an update invalidates all memory cells storing sums containing the old value of $A[i]$. For the case when updates have the form $A[i] \leftarrow A[i] + \Delta$, Yao [Yao85] proved a lower bound of $\Omega(\lg n / \lg \lg n)$. Finally, Hampapuram and Fredman [HF98] proved an $\Omega(\lg n)$ lower bound for this version of the problem; their bound holds even for the offline problem. In higher dimensions, Chazelle [Cha97] gives a lower bound of $\Omega((\lg n / \lg \lg n)^d)$, which also holds even for the offline problem.

In the group model, the best previous lower bound of $\Omega(\lg n / \lg \lg n)$ is by Fredman and Saks [FS89]. A tight logarithmic bound (including the lead constant) was given by [Fre82] for the restricted class of “oblivious” algorithms, whose behavior can be described by matrix multiplication. For the offline problem, Chazelle [Cha97] gives a lower bound of $\Omega(\lg \lg n)$ per operation; this is exponentially weaker than the best known upper bound. No better lower bounds are known in higher dimensions.

3.3. Upper bounds for the partial-sums problem. An easy $O(\lg n)$ upper bound for partial sums is to maintain a balanced binary tree with the elements of A in the leaves, augmented to store partial sums for each subtree. A

simple variation of this scheme yields an implicit data structure occupying exactly n memory locations [Fen94]. For the **SUM** query, it is easy to obtain good trade-offs. Using trees with branching factor B , one can obtain $t_q = O(\log_B n)$ and $t_u = O(B \log_B n)$, or $t_q = O(B \log_B n)$ and $t_u = O(\log_B n)$. These bounds can be rewritten as $t_q \lg \frac{t_u}{t_q} = O(\lg n)$, or $t_u \lg \frac{t_u}{t_q} = O(\lg n)$, respectively, which matches our lower bound for the case $\delta = \Theta(b)$ and for the group model. For **SELECT** queries, one cannot expect to achieve the same trade-offs because, even for a polynomial update time, there is a superconstant lower bound on the predecessor problem [BF02]. Exactly what trade-offs are possible remains an open problem.

Dietz [Die89] considers the partial-sums problem with **SUM** queries on a RAM, when $\delta = o(b)$. He achieves $O(\lg n / \lg \lg n)$ running times provided that $\delta = O(\lg \lg n)$. Raman, Raman, and Rao [RRR01] show how to support **SELECT** in $O(\lg n / \lg \lg n)$, again if $\delta = O(\lg \lg n)$. For $t_u = \Omega(\lg n / \lg \lg n)$, the same δ , and **SUM** queries, they give a trade-off of $t_q = O(\log_{t_u} n)$. They achieve the same trade-off for **SELECT** queries, when $\delta = 1$. Hon, Sadakane, and Sung [HSS03] generalize the trade-off for **SELECT** when $\delta = O(\lg \lg n)$. All of these results do not scale well with b or δ because of their use of precomputed tables.

3.4. Upper bounds for dynamic connectivity. For forests, Sleator and Tarjan's classic data structure for dynamic trees [ST83] achieves an $O(\lg n)$ upper bound for dynamic connectivity. A simpler solution is given by Euler tour trees [HK99]. This data structure can achieve a running time of $t_q = O(\frac{\lg n}{\lg(t_u/t_q)})$, matching our lower bound.

For general graphs, the first to achieve polylogarithmic time per operation were Henzinger and King [HK99]. They achieve $O(\lg^3 n)$ per update, and $O(\lg n / \lg \lg n)$ per query, using randomization and amortization. Henzinger and Thorup [HT97] improve the update bound to $O(\lg^2 n)$. Holm, de Lichtenberg, and Thorup [HdLT01] give a simple deterministic solution with the same amortized running time: $O(\lg^2 n)$ per update and $O(\lg n / \lg \lg n)$ per query. The best known result in terms of updates is by Thorup [Tho00], achieving nearly logarithmic running times: $O(\lg n (\lg \lg n)^3)$ per update and $O(\lg n / \lg \lg \lg n)$ per query. This solution is only a factor of $(\lg \lg n)^3$ away from our lower bound. Interestingly, all of these solutions are on our trade-off curve. In fact, for any $t_u = \Omega(\lg n (\lg \lg n)^3)$, Thorup's solution can achieve $t_q = O(\frac{\lg n}{\lg(t_u/t_q)})$, showing that our trade-off curve is optimal for this range of t_u .

For plane graphs, Eppstein et al. [EIT⁺92] give a logarithmic upper bound. Plane graphs are planar graphs with a given topological planar embedding, specified by the order of the edges around each vertex. Our lower bound holds for such graphs, proving the optimality of this data structure.

4. Models. The cell-probe model is a nonuniform model of computation. The memory is represented by a collection of cells. Operations are handled by an algorithm which can read and write cells from the memory; all computation is free, and the internal state is unbounded. However, the state is lost at the end of an operation. Because the state is not bounded, it can be assumed that all writes happen at the end of the operation. If cells have b bits, we restrict the number of cells to 2^b , ensuring that a pointer can be represented in one cell. This restriction is a version of the standard *transdichotomous assumption* frequently made in the context of the word RAM, and is therefore natural in the cell-probe model as well.

We extend the model to allow for nondeterministic computation, in the spirit of [HR03]. Boolean queries can spawn any number of independent execution threads;

the overall result is an accept (“yes” answer) if and only if at least one thread accepts. The running time of the operation is the running time of the longest thread. Rejecting threads may not write any cells; accepting threads may, as long as all accepting threads write exactly the same values. Because of this restriction, and because updates are deterministic, the state of the data structure is always well defined.

All lower bounds in this paper hold under Las Vegas randomization, i.e., zero-error randomization. We consider a model of randomization that is particularly easy to reason about in the case of data structures. When the data structure is created, a fixed subset of, say, 2^{b-1} cells is initialized to uniformly random values; from that point on, everything is deterministic. This model can easily simulate other models of randomization, as long as the total running time is at most 2^{b-1} (which is always the case in our lower bounds); the idea is that the data structure maintains a pointer to the next random cell and increments the pointer upon use. For nondeterministic computation, all accepting threads increment the pointer by the largest number of coins that could be used by a thread (bounded by the running time). Using this model, one can immediately apply the easy direction of Yao’s minimax principle [Yao77]. Thus, for any given distribution of the inputs, there is a setting of the random coins such that the amortized running time, in expectation over the inputs, is the same as the original algorithm, in expectation over the random coins. Using the nonuniformity in the model, we can hardwire the fixed setting of the coins into the algorithm.

We also consider Monte Carlo randomization, i.e., randomization with a two-sided error. Random coins are obtained in the same way, but now the data structure is allowed to make mistakes. We do not allow the data structure to be nondeterministic. In this paper, we are concerned only with error probabilities of $n^{-\Omega(1)}$; that is, the data structure should be correct with high probability. Note that by holding a constant number of copies of the data structure and using independent coins, we can increase the exponent of n to any desired constant. In the data-structures world, it is natural to require that data structures be correct with high probability, as opposed to the bounded-error restriction that is usually considered in complexity theory. This is because we want to guarantee correctness over a large sequence of operations. In addition, boosting the error from constant to n^{-c} requires $O(\lg n)$ repetitions, which is usually not significant for an algorithm, but is a significant factor in the running time of a data-structure operation.

5. Lower bounds, take one. In this section, we give the intuition behind our approach and detail a simple form of it that allows us to prove an $\Omega(\lg n)$ lower bound on the partial-sums problem when $\delta = \Theta(b)$, which is tight in this case. This proof serves as a warmup for the more complicated results in sections 6 and 7.

5.1. General framework. We begin with the framework for our lower bounds in general terms. Consider a sequence of data-structure operations A_1, A_2, \dots, A_m , where each A_i incorporates all information characterizing operation i , i.e., the operation type and any parameters for that type of operation. Upon receiving request A_i , the data structure must produce an appropriate response. The information gathered by the algorithm during a query (by probing certain cells) must uniquely identify the correct answer to the query, and thus must encode sufficient information to do so.

To establish the lower bounds of this paper, we establish lower bounds for a simpler type of problem. Consider two adjacent intervals of operations: A_i, \dots, A_{j-1} and A_j, \dots, A_k . At all times, conceptually associate with each memory cell a *chronogram* [FS89], i.e., the index t of the operation A_t during which the memory cell was last modified. Now consider all read instructions executed by the data structure during

operations A_j, \dots, A_k that access cells with a chronogram in the interval $[i, j - 1]$. In other words, we consider the set of cells written during the time interval $[i, j - 1]$ and read during the interval $[j, k]$ before they are overwritten. All *information transfer* from time interval $[i, j - 1]$ to time interval $[j, k]$ must be encoded in such cells and must be executed by such cell writes and reads. If the queries from the interval $[j, k]$ depend on updates from the interval $[i, j - 1]$, all the information characterizing this dependency must come from these cell probes, because an update happening during $[i, j - 1]$ cannot be reflected in a cell written before time i . The main technical part of our proofs is to establish a lower bound on the amount of information that must be transferred between two time intervals, which implies a corresponding lower bound on the number of cells that must be written and read to execute such a transfer. Such bounds will stem from an encoding argument, in conjunction with a simple information-theoretic analysis.

Next we show how to use such a lower bound on the information transfer between two adjacent intervals of operations to prove a lower bound on the data-structure problems we consider. Consider a binary tree whose leaves represent the entire sequence of operations in time order. Each node in the tree has an associated time interval of operations, corresponding to the subtree rooted at that node. We can obtain two adjacent intervals of operations by, for example, considering the two nodes with a common parent. For every node in the tree, we define the *information transfer through that node* to be the number of read instructions executed in the subtree of the node's right child that read data written by (i.e., cells last written by) operations in the subtree of the node's left child. The lower bound described above provides a lower bound on this information transfer for every node. We combine these bounds into a lower bound on the number of cell probes performed during the entire execution by simply summing over all nodes.

To show that this sum of individual lower bounds is indeed an overall lower bound, we make two important points. First, we claim that we are not double counting any read instructions. Any read instruction is characterized by the time when it occurs and the time when the location was last written. Such a read instruction is counted by only one node, namely, the lowest common ancestor of the read and write times, because the write must happen in the left subtree of the node, and the read must happen in the right subtree. The second point concerns the correctness of summing up individual lower bounds. This approach works for the arguments in this paper, because all lower bounds hold in the average case under the same probability distribution for the operations. Therefore, we can use linearity of expectation to break up the total number of read instructions performed on average into these distinct components. Needless to say, worst-case lower bounds could not be summed in this way.

The fact that our lower bounds hold in the average case of an input distribution has another advantage: the same lower bound holds in the presence of Las Vegas randomization. The proofs naturally allow the running time to be a random variable, depending on the input. By the easy direction of the minimax principle, a Las Vegas randomized data structure can be converted into a deterministic data structure that, on a given random distribution of the inputs, achieves the same expected running time.

This line of argument has an important generalization that we use for proving trade-off lower bounds. Instead of considering a binary tree, we can consider a tree of arbitrary degree. Then we may consider the information transfer either between any node and all its left siblings, or between any node and all its right siblings. Neither of these strategies double counts read instructions, because a read instruction is counted

only for a node immediately below the lowest common ancestor of the read and write times.

5.2. An initial bound for the partial-sums problem. We are now ready to describe an initial lower bound for the partial-sums problem, which gives a clear and concise materialization of the general approach from the previous section. We will prove a lower bound of $\Omega(\frac{\delta}{b} \lg n)$, which is tight (logarithmic) for the special case of $\delta = \Theta(b)$. The bound from this section considers only `SUM` queries and does not allow nondeterminism.

It will be useful to analyze the partial-sums problem over an arbitrary group with at least 2^δ elements. Our proof will not use any knowledge about the group, except the quantity δ . Naturally, the data structure is allowed to know the group; in fact, the data structure need only work for one arbitrary choice of group. In particular, the lower bound will hold for the group $\mathbb{Z}/2^\delta\mathbb{Z}$, the group of δ -bit integers with addition modulo 2^δ . A solution to the original partial-sums problem also gives a solution to the problem over this group, as long as we can avoid overflowing a cell in the original problem. To guarantee this, it suffices that $\delta + \lg n < b$. By definition of the model, we always have $\lg n \leq b$ and $\delta \leq b$, so we can avoid overflow by changing only constant factors.

We consider a sequence of $m = \Omega(\sqrt[3]{n})$ operations, where m is a power of 2. Operations alternate between updates and queries. We choose the index in the array touched by the operation uniformly at random. If the operation is an update, we also choose the value Δ uniformly at random. This notion of random updates and queries remains unchanged in our subsequent lower bounds, but the pattern of alternating updates and queries changes. Our lemmas do not assume anything about which operations are updates or queries, making it possible to reuse them later.

Our lower bound is based on the following lemma analyzing intervals of operations.

LEMMA 5.1. *Consider two adjacent intervals of operations such that the left interval contains L updates, the right interval contains L queries, and overall the intervals contain $O(\sqrt[3]{n})$ operations. Let c be the number of read instructions executed during the second interval that read cells last written during the first interval. Then $E[c] = \Omega(\frac{\delta}{b}L)$.*

Before we embark on a proof of the lemma, we show how it implies our logarithmic lower bound. As in the framework discussion, we consider a complete binary tree with one leaf per operation. For every node v , we analyze the information transfer through v , i.e., the read instructions executed in the subtree of v 's right child that access cells with a chronogram in the subtree of v 's left child. If v is on the $\frac{1}{3} \lg n$ bottommost levels, the conditions of the lemma are satisfied, with L being a quarter of the number of leaves under v . Then, the information transfer through v is $\Omega(L\frac{\delta}{b})$ on average. As explained in the framework discussion, we can simply sum these bounds for all nodes to get a lower bound for the execution time. The information transfer through all nodes on a single level is $\Omega(m\frac{\delta}{b})$ in expectation (because these subtrees are disjoint). Over $\frac{1}{3} \lg n$ levels, the lower bound is $\Omega(m\frac{\delta}{b} \lg n)$, or amortized $\Omega(\frac{\delta}{b} \lg n)$ per operation.

5.3. Interleaving between two intervals. The lower bound for two adjacent intervals of operations depends on the interleaving between the indices updated and queried in the two intervals. More precisely, we care about the indices a_1, a_2, \dots touched by updates during the left interval of time, and the indices b_1, b_2, \dots queried during the right interval. By relabeling, assume that $a_1 \leq a_2 \leq \dots$ and $b_1 \leq b_2 \leq \dots$.

We define the *interleaving number* l to be the number of indices i such that, for some index j , $a_i < b_j \leq a_{i+1}$. In other words, the interleaving number counts transitions from runs of a 's to runs of b 's when merging the two sorted lists of indices.

LEMMA 5.2. *Consider two adjacent intervals of operations such that the left interval contains L updates, the right interval contains L queries, and overall the intervals contain $O(\sqrt[3]{n})$ operations. Then the interleaving between the two intervals satisfies $E[l] = \Theta(L)$ and, with probability $1 - o(1)$, no index is touched by more than one operation.*

Proof. By the birthday paradox, the expected number of indices touched more than once is at most $O((\sqrt[3]{n})^2) \cdot \frac{1}{n} = O(n^{-1/3})$. By Markov's inequality, all indices are unique with probability $1 - O(n^{-1/3})$. Because $l \leq L$, it suffices to prove the lower bound. We show $E[l \mid \text{all indices are unique}] = \Omega(L)$. Because the condition is met with $\Omega(1)$ probability, $E[l] = \Omega(L)$. Fix the set S of $2L$ relevant indices arbitrarily. It remains to randomly designate L of these to be updates from the left interval, and the rest of S to be queries from the right interval. Then l is the number of transitions from updates to queries, as we read S in order. The probability that a transition happens on any fixed position is $\frac{1}{4}$, so by linearity of expectation, $E[l \mid S] = \Omega(L)$. Because this bound holds for any S , we can remove the conditioning. \square

The following information-theoretic lemma will be used throughout the paper by comparing the lower bound it gives with upper bounds given by various encoding algorithms. For an introduction to information theory, we refer the reader to [CT91]. Remember that we are considering the partial-sums problem over an arbitrary group with at least 2^δ elements.

LEMMA 5.3. *Consider two adjacent intervals of operations such that the left interval contains L updates, the right interval contains L queries, and overall the intervals contain $O(\sqrt[3]{n})$ operations. Let G be the random variable giving the indices touched by every operation, and giving the Δ values for all updates except those in the left interval. Let S be the random variable giving all partial sums queried in the right interval. Then $H(S \mid G) = \Omega(L\delta)$.*

Proof. Fix $G = g$ to an arbitrary value such that no index is touched twice in the two intervals. Let l be the interleaving between the two intervals (l is a function of g). Let U denote the set of indices updated in the left interval. By the definition of the interleaving number, there must exist l queries in the right interval to indices $q_1 < q_2 < \dots < q_l$ such that $U \cap [q_{t-1} + 1, q_t] \neq \emptyset$ for each $t \geq 1$, where q_0 is taken to be $-\infty$. Now let us consider the partial sums queried by these l queries, which we denote S_1, S_2, \dots, S_l . The terms of these sums are elements of the array $A[1..n]$ at the time the query is made. Some elements were set by updates before the first interval, or during the second interval, so they are constants for $G = g$. However, each S_t contains a random term in $[q_{t-1} + 1, q_t]$, which comes from an update from the first interval. This element was not overwritten by a fixed update from the second interval because, by assumption, no index was updated twice. Then each S_t will be a random variable uniformly distributed in the group: even if we condition on arbitrary values for all but one of the random terms, the sum remains uniformly random in the group because of the existence of inverses. Furthermore, the random variables will be independent, because S_t contains at least one random term that was not present in any S_r with $r < t$ (namely, the term in $[q_{t-1} + 1, q_t]$). Then $H((S_1, \dots, S_l) \mid G = g) = l\delta$. The variable S entails S_1, \dots, S_l , so $H(S \mid G = g) \geq l\delta$. By Lemma 5.2, $E[l] = \Omega(L)$. Furthermore, with probability $1 - o(1)$, a random G leads to no index being updated twice in the two intervals, so the above analysis applies. Then $H(S \mid G) = \Omega(L\delta)$. \square

5.4. Proof of Lemma 5.1. We consider two adjacent intervals of time, the first spanning operations $[i, j - 1]$ and the second spanning operations $[j, k]$. We propose an encoding for the partial sums queried in $[j, k]$ given the value of G and compare its size to the $\Omega(L\delta)$ lower bound of Lemma 5.3. Our encoding is simply the list of addresses and contents of the cells probed in the right interval that were written in the left interval. Thus, we are proposing an encoding of expected size $E[c] \cdot 2b$ bits, proving that $E[c] = \Omega(L\frac{\delta}{b})$. It should be noted that c is a random variable, because the algorithm can make different cell probes for different update parameters.

To recover the partial sums from this encoding, we begin by running the algorithm for the time period $[1, i - 1]$; this is possible because all operations before time i are known given G . We then skip the time period $[i, j - 1]$ and run the algorithm for the time period $[j, k]$, which will return the partial sums queried during this time. To see why this is possible, notice that a read instruction issued during time period $[j, k]$ falls into one of the following three categories, depending on the time t_w when the cell was written:

- $t_w \geq j$: We can recognize this case by maintaining a list of memory locations written during the simulation; the data is immediately available.
- $i \leq t_w < j$: The contents of the memory location are available as part of our encoding; we can recognize this case by examining the set of addresses in the encoding.
- $t_w < i$: This is the default case, if we failed to satisfy the previous conditions. The contents of the cell are determined from the state of the memory upon finishing the first simulation up to time $i - 1$.

5.5. Obtaining trade-off lower bounds. We now show how our framework can be used to derive trade-off lower bounds. In a nutshell, we consider instances when the cheaper operation is performed more frequently, so that the total cost of queries matches the total cost of updates. Then, we analyze the sequence of operations by considering a tree with a higher branching factor.

Assume there exists a data structure with amortized expected running times bounded by t_u for updates and t_q for queries. Our hard instance consists of blocks of $t_u + t_q$ operations. Each block contains t_q updates and t_u queries; the order inside a block is irrelevant. We generate the arguments to updates and queries randomly as before. Let $B = 2 \cdot \max\{\frac{t_u}{t_q}, \frac{t_q}{t_u}\}$. We prove below that the expected amortized cost of a block is $\Omega(\max\{t_u, t_q\} \frac{\delta}{b} \log_B n)$. On the other hand, the expected amortized cost of a block is at most $2t_u t_q$. This implies $\frac{t_u t_q}{\max\{t_u, t_q\}} = \Omega(\frac{\delta}{b} \log_B n)$, so $\min\{t_u, t_q\} \cdot \lg \frac{\max\{t_u, t_q\}}{\min\{t_u, t_q\}} = \Omega(\frac{\delta}{b} \lg n)$. This is the desired trade-off, which is tight when $\delta = \Theta(b)$.

To prove the lower bound on blocks, consider a balanced B -ary tree in which the leaves correspond to blocks. We let the total number of blocks be $m = \Theta(\sqrt[b]{n})$. If $\max\{t_u, t_q\} = \Omega(\sqrt[b]{n})$, our lower bound states that $\min\{t_u, t_q\} = \Omega(1)$, so there is nothing to prove. Thus, we can assume $t_u + t_q = O(\sqrt[b]{n})$, which bounds the number of operations in a block. Then, the total number of operations is $O(\sqrt[b]{n})$, satisfying one of the conditions of Lemma 5.1.

For the case $t_u \geq t_q$, we are interested in the information transfer between each node and its left siblings. The subtree of the node defines the right interval of operations, and the union of the subtrees of all left siblings defines the left interval. Let L be the number of blocks in the right interval. We make a claim only regarding nodes that are in the right half of their parent's children. In this case, the number of blocks in the left interval is at least $\frac{B}{2}L$. Then, the number of queries in the right interval

is Lt_u , while the number of updates in the left interval is at least $L \frac{t_u}{t_q} t_q = Lt_u$. We can then apply Lemma 5.1; having more updates in the left interval cannot decrease the bound, because moving the beginning of the left interval earlier can only increase the number of cell probes that are counted. Therefore, the expected number of cell probes associated with this node is $\Omega(Lt_u \frac{\delta}{b})$. Now we sum the lower bounds for all nodes on a level and obtain that the number of cell probes associated with that level is $\Omega(mt_u \frac{\delta}{b})$. Summing over all levels, we get an amortized lower bound of $\Omega(t_u \frac{\delta}{b} \log_B n)$ per block, as desired.

For the case $t_u < t_q$, we apply a symmetric argument. We analyze the information transfer between any node, giving the left interval, and all its right siblings, giving the right interval. For nodes in the first half of their parent's children, the left interval contains Lt_q updates, while the right interval contains at least Lt_q queries. By Lemma 5.1, the expected number of cell probes associated with this node is $\Omega(Lt_q \frac{\delta}{b})$. Thus, the number of cell probes associated with a level is $\Omega(mt_q \frac{\delta}{b})$, and the amortized bound per block is $\Omega(t_q \frac{\delta}{b} \log_B n)$.

5.6. Refinements. First, note that our lower bounds so far depend only on randomness in the update parameters Δ , and not on randomness in the update or query indices. Indeed, the value of G is irrelevant, except for the interleaving number that it yields. It follows that the logarithmic lower bound and the trade-off lower bound are also true for sequences of operations in which we fix everything except the Δ parameters, as long as such sequences have a high sum of the interleaving numbers of each node. Our application of Lemma 5.2 can be seen as a probabilistic proof that such bad sequences exist.

The prototypical deterministic sequence with high total interleaving is the bit-reversal permutation. For any n that is a power of 2, consider the permutation $\pi : \{0, \dots, n-1\} \rightarrow \{0, \dots, n-1\}$ that takes i to the integer obtained by reversing i 's $\log_2 n$ bits. The corresponding access sequence consists of n pairs of **UPDATE** and **SUM**, the i th pair touching index $\pi(i)$. The bit-reversal permutation underlies the fast Fourier transform algorithm. It also gives an access sequence that takes $\Omega(\lg n)$ amortized time for any binary search tree [Wil89]. Finally, it was used to prove an $\Omega(\lg n)$ bound for the partial-sums problem in the semigroup model [HF98]. To see why this permutation has high total interleaving, consider the following recursive construction. The permutation π' of order $2n$ is obtained from a permutation π of order n by the rules $\pi'(i) = 2 \cdot \pi(i)$, $\pi'(i+n) = 2 \cdot \pi(i) + 1$, for $i \in \{0, 1, \dots, n-1\}$. Each level of the recursion adds an interleaving of n between the left and right halves, so the total interleaving is $\Theta(n \lg n)$.

The fact that our lower bound holds for fixed sequences of operations implies the same lower bound in the group model. A solution in the group model handles every **UPDATE** and **SUM** by executing a sequence of additions on cells containing abstract elements from the group. The cells touched by these additions depend only on the indices touched by queries and updates, because the data structure treats the group as a black box and cannot examine the Δ 's. So if we know a priori the sequence of indices touched by queries and updates, we can implement the same solution in the cell-probe model for the group $\mathbb{Z}/2^b\mathbb{Z}$; because the Δ 's are unrestricted elements of the group, $\delta = b$. The group additions can be hardwired into our solution for the cell-probe model through nonuniformity, and cell probes are needed only to execute the actual additions.

5.7. Duality of lower and upper bounds. Recall the classic upper bound for the partial-sums problem. We maintain a tree storing the elements of the array in

order in the leaves. Each node stores the sum of all leaves in its subtree. An update adds Δ to the subtree sums along the root-to-leaf path of the touched element. A query traverses the root-to-leaf path of the element and reports the sum of all subtrees to the left of the path.

Our lower bound can be seen as a dual of this natural algorithm. To see this, we describe what happens when we apply the lower bound analysis to the algorithm. We argue informally. Consider two intervals of 2^k operations. The information transfer between the intervals is associated with a node of height k in the lower bound tree. On the other hand, the indices of the operations will form a relatively uniformly spaced set of $O(2^k)$ indices. Thus, the distance in index space between a query from the right interval and the closest update from the left interval will usually be around $n/2^k$. The algorithm's tree passes information between the update and the query through the lowest common ancestor of the two indices. Because of the separation between the indices, this will usually be a node at level around $\lg n - k$. Thus, we can say that our lower bound is roughly an upside-down view of the upper bound. The information passed through the k th level from the bottom of one tree is roughly associated with the k th level from the top of the other tree.

6. Handling queries with low output entropy. The lower bound technique as presented so far depends crucially on the query answers having high entropy: the information transfer through a node is bounded from below by the entropy of all queries from the right subtree of the node. However, in order to prove lower bounds for dynamic language membership problems (such as dynamic connectivity), we need to be able to handle queries with binary answers. To prove lower bounds for a pair of adjacent intervals, it is tempting to consider the communication complexity between a party holding the updates from the left interval and a party holding the queries from the right interval. Many bounds for communication complexity hold even for decision problems, so queries with binary output should not be a problem. However, a solution for the data structure does not really translate well into the communication-complexity setting. The query algorithm probes many cells, only a few of which (a logarithmic fraction) are in the left interval. If the party with the right interval communicates all these addresses, just to get back the answer “not written in the left interval” for most of them, the communication complexity blows up considerably. One could also imagine a solution based on approximate dictionaries, where the party holding the left interval sends a sketch of the cells that were written, allowing the other party to eliminate most of the uninteresting cell probes. However, classic lower bounds for approximate dictionaries [CFG⁺78] show that it is impossible to send a sketch that is small enough for our purposes. The solution developed in this section is not based on communication complexity, although it can be rephrased in terms of nondeterministic communication complexity. While this solution is not particularly hard, we find it to be quite subtle.

6.1. Setup for the lower bound. Our approach is to construct hard sequences of operations that will have a fixed response, and the data structure need only confirm that the answer is correct. Such predictable answers do not trivialize the problem: the data structure has no guarantee about the sequence of operations, and the information it gathers during a query (by probing certain cells) must provide a certificate stating that the predicted answer is correct. In other words, the probed cells must uniquely identify the answer to the query, and thus must encode sufficient information to do so. As a consequence, our lower bounds hold even if the algorithm makes nondeterministic cell probes, or if an all-powerful prover reveals a minimal set of cells sufficient to show

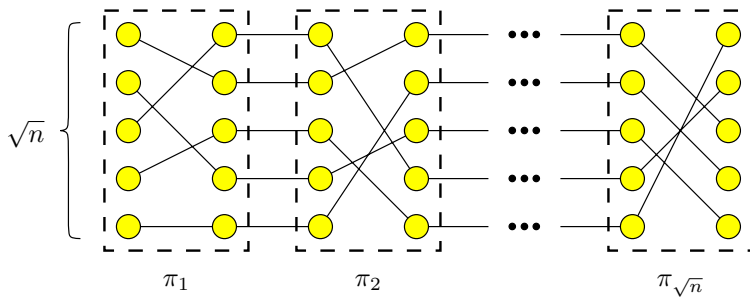


FIG. 6.1. Our graphs can be viewed as a sequence of permutation boxes (dashed). The horizontal edges between boxes are in fact contracted in the actual graphs.

that a certain answer to a query is correct.

The machinery developed in this section is also necessary in the case of the partial-sums problem if we want a lower bound for sequences of **UPDATE** and **SELECT** operations. Even though **SELECT** returns an index in the array, i.e., $\lg n$ bits, it is not clear how more than one bit of information can be used for a lower bound argument. Instead, we consider a **VERIFY-SUM** operation, which is given a sum Σ and an index i , and tests whether the partial sum up to i is equal to Σ . In principle, this operation can be implemented by two calls to **SELECT**, namely, by testing that $i = \text{SELECT}(\Sigma) = \text{SELECT}(\Sigma - 1) + 1$.

Below we give a single lower bound proof that applies to both the partial-sums problem with verify and dynamic connectivity. We accomplish this by giving a proof for the partial-sums problem over any group G with at least 2^δ elements and then specializing G for the two problems we consider.

For the partial-sums problem with **SELECT**, we use $G = \mathbb{Z}/2^\delta\mathbb{Z}$. This introduces a slight complication, because **VERIFY-SUM** in modulo arithmetic can no longer be implemented by a constant number of calls to **SELECT**. To work around this issue, remember that our lower bound for **VERIFY-SUM** also holds for nondeterministic computation. To implement **VERIFY-SUM**(i, Σ) nondeterministically, we guess a b -bit quantity Σ' such that $\Sigma' \bmod 2^\delta = \Sigma$, and verify the old condition $i = \text{SELECT}(\Sigma') = \text{SELECT}(\Sigma' - 1) + 1$. We have implicitly assumed that **SELECT** is deterministic, which is natural because **SELECT** does not return a binary answer. Note that only one thread accepts, so there is no problem if **SELECT** updates memory cells (the updates made by the sole accepting thread are the ones that matter).

For the dynamic-connectivity problem, we use $G = S_{\sqrt{n}}$, i.e., the permutation group on \sqrt{n} elements. Notice that now we have $\delta = \sqrt{n} \lg \sqrt{n} - \Theta(\sqrt{n})$, a very large quantity, unlike in the partial-sums problem, where it was implied that $\delta < b$. Our proof never actually assumes any particular relation between δ and b .

To understand the relation between this problem and dynamic connectivity, refer to Figure 6.1. We consider a graph whose vertices form an integer grid of size \sqrt{n} by \sqrt{n} . Edges only connect vertices from adjacent columns. Each vertex is incident to at most two edges, one edge connecting to a vertex in the previous column and one edge connecting to a vertex in the next column. These edges do not exist only when the vertex is in the first or last column. The edges between two adjacent columns of vertices thus form a perfect matching in the complete bipartite graph $K_{\sqrt{n}, \sqrt{n}}$, describing a permutation of order \sqrt{n} . More precisely, point (x, y_1) in the grid is connected to point $(x + 1, y_2)$ exactly when $\pi_x(y_1) = y_2$ for a permutation π_x .

Another way to look at the graph is in terms of permutation networks. We can imagine that the graph is formed by \sqrt{n} horizontal wires, going between permutation boxes. Inside each box, the order of all wires is changed arbitrarily.

Our graph is always the disjoint union of \sqrt{n} paths. This property immediately implies that the graph is plane, because any embedding maintains planarity (though the edges may have to be routed along paths with several bends).

The operations required by the partial-sums problem need to be implemented in terms of many elementary operations, so they are actually “macro-operations.” Macro-operations are of two types, **UPDATE** and **VERIFY-SUM**, and all receive as parameters a permutation and the index x of a permutation box. To perform an update, all the edges inside the named permutation box are first deleted and then reconstructed according to the new permutation. This translates to \sqrt{n} **DELETE**’s and \sqrt{n} **INSERT**’s in the dynamic-connectivity world. Queries on box x test that point $(1, y)$ is connected to point $(x + 1, \pi(y))$ for all $y \in \{1, 2, \dots, \sqrt{n}\}$. This requires \sqrt{n} connectivity queries. The conjunction of these tests is equivalent to testing that the composition of $\pi_1, \pi_2, \dots, \pi_x$ (the permutations describing the boxes to the left) is identical to the given permutation π —the **VERIFY-SUM** in the partial-sums world.

As stated before, the lower bound we obtain is $\Omega(\frac{\delta}{b} \lg n)$. For dynamic connectivity, we are interested in $b = \Theta(\lg n)$, which is the natural word size for this problem. As we saw already, $\delta = \Theta(\sqrt{n} \lg n)$. Thus, our lower bound translates into $\Omega(\sqrt{n} \lg n)$. This is a lower bound for the macro-operations, though, which are implemented through $O(\sqrt{n})$ elementary operations. Therefore, the lower bound for dynamic connectivity is $\Omega(\lg n)$, as desired. The same calculation applies to the trade-off expressions, which essentially means that the $\frac{\delta}{b}$ term should be dropped to obtain the true bound for dynamic connectivity.

6.2. Proof of the lower bound. As before, the sequence of operations alternates between **UPDATE** and **VERIFY-SUM**. The index queried or updated is chosen uniformly at random. If the operation is an **UPDATE**, we select a random element of G for the value Δ . If the operation is **VERIFY-SUM**, we give it the composition of the elements before the queried index. This means that the data structure will be asked to prove a tautology, involving the partial sum up to that index.

Because of this construction of the hard sequence, at least one nondeterministic thread for each query should accept. For every random input, let us fix one accepting thread for each operation. When we mention cells that are “read,” we mean cells read in this chosen execution path; by definition of the model, writes are the same for all accepting threads. As in the framework discussion, we are interested in lower bounds for the information transfer between adjacent intervals of operations. The following lemma is an analogue of Lemma 5.1.

LEMMA 6.1. *Consider two adjacent intervals of operations such that the left interval contains L updates, the right interval contains L queries, and overall the intervals contain $O(\sqrt[3]{n})$ operations. Let w be the number of write instructions executed during the first interval, and let r be the number of read instructions executed during the second interval. Also, let c be the number of read instructions executed during the second interval that read cells last written during the first interval. Then $E[c] = \Omega(L \frac{\delta}{b}) - O(\frac{E[s]}{b})$, where $s = \lg \binom{r+w}{r}$.*

Note that the lower bound of this lemma is weaker than that of Lemma 5.1, because of the additional term $\frac{E[s]}{b}$. Before we prove this lemma (in the next section), let us show that this term is inconsequential, and the lemma implies the same bounds and trade-offs.

Consider a sequence of $m = \Theta(\sqrt[3]{n})$ operations, and let T be the total running time of the data structure. We construct a complete binary tree over these operations. Consider a node v that is a right child, and let L be the number of leaves in its subtree. By Lemma 6.1, we have $E[c] = \Omega(L \frac{\delta}{b}) - O(\frac{E[s]}{b})$, where c is the number of cell probes associated with v . Note that $s = \lg \binom{r+w}{r} \leq r + w$; thus, s is bounded by the number of read instructions in v 's subtree, plus the number of write instructions in the subtree of v 's left sibling. Summing for all nodes on a level, s counts all read and write instructions at most once, so we obtain $E[\sum c_i] = \Omega(m \cdot \frac{\delta}{b}) - O(\frac{E[T]}{b})$. We sum up the lower bounds for each level to obtain a lower bound on $E[T]$. We obtain that $E[T] = \Omega(m \cdot \frac{\delta}{b} \lg n) - O(\lg n \cdot \frac{E[T]}{b})$. Because $\lg n \leq b$, this means that $E[T] = \Omega(m \cdot \frac{\delta}{b} \lg n)$. This result implies an average-case amortized lower bound per operation of $\Omega(\frac{\delta}{b} \lg n)$.

To obtain trade-off lower bounds, we apply the same reasoning as in section 5.5. The only thing we have to do is verify that the new term depending on $E[s]$ does not affect the end result. When we sum the lower bounds for one level in the tree, we lose a term of $O(\sum \frac{E[s_i]}{b})$, compared to the old bound. Here i ranges over all nodes at that level. We must understand $\sum s_i = \lg \prod \binom{r_i+w_i}{r_i}$ in terms of T , the total running time for the entire sequence of operations.

The quantity $\prod \binom{r_i+w_i}{r_i}$ counts the total numbers of ways to choose r_i elements from a set $r_i + w_i$, where we have a different set for each i . This is bounded from above by the number of ways to choose $\sum r_i$ elements from a single set of $\sum (r_i + w_i)$ objects. Thus, $\sum s_i \leq \lg \binom{\sum (r_i+w_i)}{\sum r_i}$. Assume we have upper bounds $\sum r_i \leq U_r$ and $\sum w_i \leq U_w$. Then, we can write $\binom{\sum (r_i+w_i)}{\sum r_i} \leq \binom{U_r+U_w}{\sum r_i} \leq \binom{2(U_r+U_w)}{\sum r_i} \leq \binom{2(U_r+U_w)}{U_r}$. The last inequality holds because $\binom{n}{k}$ increases with k for $k \leq \frac{n}{2}$. We entered this regime by artificially doubling $U_r + U_w$. Because r_i and w_i are symmetric, we also have $\binom{\sum (r_i+w_i)}{\sum r_i} = \binom{\sum (r_i+w_i)}{\sum w_i} \leq \binom{2(U_r+U_w)}{U_r}$.

Now we need to develop the upper bounds U_r and U_w . For the case $t_u \leq t_q$, our proof considered intervals formed by a node and all its left siblings. Thus, $\sum r_i$ counts each read instruction once, for the node it is under; so $\sum r_i \leq T$. On the other hand, $\sum w_i \leq B \cdot T$, because a write instruction is counted for every right sibling of its ancestor on the current level. For the case $t_u > t_q$, we consider intervals formed by a node and all its right siblings. Thus, $\sum w_i \leq T$ and $\sum r_i \leq B \cdot T$.

Using these bounds, we see that $\sum s_i \leq \lg \binom{2(B+1)T}{T} = O(T \lg B)$. Because this upper bound holds in any random instance, it also holds in expectation: $E[\sum s_i] = O(E[T] \lg B)$. So our lower bound loses $O(\frac{E[T] \lg B}{b})$ per level, which, over all levels, sums to $O(\frac{E[T] \lg B}{b} \log_B n) = O(E[T] \frac{\lg n}{b})$. Because $\lg n \leq b$, our lower bound on $E[T]$ is equal to the old lower bound minus $O(E[T])$. Thus, we lose only a constant factor in the lower bound, and the results of section 5.5 continue to hold.

6.3. Proof of Lemma 6.1. The proof is an encoding argument, which is similar in spirit to the proof of Lemma 5.1 but requires a few significant new ideas. The difference from the previous proof is that the partial sums that we want to encode are no longer returned by queries, but rather they are given as parameters. Our strategy is to recover the partial sums by simulating each query for all possible parameters and see which one leads to an accept. However, these simulations may read a large number of cells, which we cannot afford in the encoding. Instead, we add a new part to the encoding which enables us to stop simulations that try to read cells we don't know. The difficulty is making this new component of the encoding small enough.

As before, we consider two adjacent intervals of operations, the first spanning $[i, j - 1]$ and the second $[j, k]$. We propose an encoding for the partial sums passed to the **VERIFY-SUM** operations during $[j, k]$, given the variable G defined in Lemma 5.3. By this lemma, such an encoding must have size $\Omega(L\delta)$ bits.

The encoder first simulates the entire execution of the data structure. Each query is given the correct partial sum, so it must accept. We choose arbitrarily one of the accepting threads. Consider the following sets of cells, based on this computation history:

- W = cells which are updated by the data structure during the interval of time $[i, j - 1]$, and never read during $[j, k]$.
- R = cells which are read by the data structure during $[j, k]$ and their last update before the read happened before time i .
- C = cells which are read by the data structure during $[j, k]$ and their last update before the read happened during $[i, j - 1]$.

These are simple sets, so, for example, cells written multiple times during $[i, j - 1]$ are included only once in W . We have $|C| = c$, $|W| \leq w$, $|R| \leq r$. Note that all of c , $|W|$, w , $|R|$, and r are random variables, because the data structure can behave differently depending on the Δ 's passed to the updates. We will give an encoding for the queried partial sums that uses $O(b) + c \cdot 2b + O(s)$ bits, where $s = \lg \binom{r+w}{r}$. Because the expected size of our encoding must be $\Omega(L\delta)$, we obtain that $E[c] + \frac{E[s]}{\Theta(b)} = \Omega(L\frac{\delta}{b})$, and therefore $E[c] = \Omega(L\frac{\delta}{b}) - O(\frac{E[s]}{b})$.

Our encoding consists of two parts. The first encodes all information about the interesting cell probes (the information transfer): for each cell in C , we encode the address of the cell and its contents at time j . This uses $O(b)$ bits to write the size of C , and $c \cdot 2b$ for the information about the cells. The second part is concerned with the “uninteresting” cell probes, i.e., those in R . This accounts for a covert information transfer: the fact that a cell was *not* written during $[i, j - 1]$ is a type of information transmitted to $[j, k]$. The part certifies that W and R are disjoint, by encoding a set S , such that $R \subset S$ and $W \subset \bar{S}$. We call S a separator between R and W . To efficiently encode a separator, we need the following result.

LEMMA 6.2. *For any integers a, b, u with $a + b \leq u$, there exists a system of sets \mathbb{S} with $\lg |\mathbb{S}| = O(\lg \lg u + \lg \binom{a+b}{a})$ such that, for all $A, B \subset \{1, 2, \dots, u\}$ with $|A| \leq a$, $|B| \leq b$, $A \cap B = \emptyset$, there exists an $S \in \mathbb{S}$ satisfying $A \subset S$ and $B \subset \bar{S}$.*

Proof. It suffices to prove the lemma for $|A| = a$ and $|B| = b$, because we can simply add some elements from $\{1, 2, \dots, u\} \setminus (A \cup B)$ to pad the sets to the right size. We use the probabilistic method to show that a good set system exists. Select a set S randomly by letting every element $x \in \{1, 2, \dots, u\}$ be in the set with probability $p = \frac{a}{a+b}$. Then, for any pair A, B , the probability that $A \subset S$ and $B \subset \bar{S}$ is $p^a(1 - p)^b$. The system \mathbb{S} will be formed of sets chosen independently at random, so the probability that there is no good S for some A and B is $(1 - p^a(1 - p)^b)^{|\mathbb{S}|} \leq \exp(-p^a(1 - p)^b|\mathbb{S}|)$. The number of choices for A and B is $\binom{u}{a} \binom{u-a}{b} \leq u^{a+b}$. So the probability that there is no good set in \mathbb{S} for any A, B is at most $u^{a+b} \exp(-p^a(1 - p)^b|\mathbb{S}|) = \exp((a + b) \ln u - p^a(1 - p)^b|\mathbb{S}|)$. As long as this probability is less than 1, such a system \mathbb{S} exists. So we want $(a + b) \ln u < p^a(1 - p)^b|\mathbb{S}| = (\frac{a}{a+b})^a (\frac{b}{a+b})^b |\mathbb{S}|$. We want to choose a system of size greater than $\frac{(a+b)^{a+b+1} \ln u}{a^a b^b}$. Then $\lg |\mathbb{S}| = \Theta((a + b + 1) \log_2(a + b) + \lg \lg u - a \log_2 a - b \log_2 b)$. Assume by symmetry that $a \leq b$. Then $\lg |\mathbb{S}| = \Theta(\lg \lg u + a \lg \frac{b}{a} + a \cdot \frac{b}{a} \log_2(1 + \frac{a}{b}))$. Let $t = \frac{b}{a}$; then $t \log_2(1 + \frac{1}{t}) = \log_2((1 + \frac{1}{t})^t) \rightarrow \log_2 e$ as $t \rightarrow \infty$. We then have

$\frac{b}{a} \log_2(1 + \frac{a}{b}) = \Theta(1)$, so our result simplifies to $\lg |\mathbb{S}| = \Theta(\lg \lg u + a \lg(b/a))$. It is well known that $a \lg(b/a) = \Theta(\lg \binom{a+b}{a})$ for $a \leq b$. \square

We apply this lemma with parameters r , w , and 2^b . First, we encode r and w , using $O(b)$ bits. The encoding and decoding algorithms can simply iterate over all possible systems \mathbb{S} for the given r and w , and choose the first good one (in the sense of the lemma). Given this unique choice of a system, a separator between R and W is the index of an appropriate set in the system. This index will occupy $O(\lg \lg(2^b) + \lg \binom{r+w}{r}) = O(\lg b + s)$ bits.

It remains to show that this information is enough to encode the sequence of queried partial sums. We simulate the data structure for the interval $[j, k]$, and prove by induction on time steps that all cell writes made by these operations are correctly determined, and all partial sums appearing in `VERIFY-SUM`'s are recovered. Updates are easy to handle, because their parameters are known given G , and they are deterministic. Thus, we can simply simulate the update algorithm. We are guaranteed that all cells that are read and have a chronogram in $[i, j-1]$ appear in C , so we can identify these cells and recover their contents. All other cells have a known content given G , so we can correctly simulate the update.

In the case of `VERIFY-SUM`, we do not actually know the sum passed to it, so we cannot simply simulate the algorithm. Instead, we try all partial sums that could be passed to the query, and for each one try all possible execution paths that the data structure can explore through nondeterminism. The cell probes made while simulating such a thread fall in one of the following cases:

- The cell was written by the data structure after time j . This case can be identified by looking at the set of cells written during the simulation. By the induction hypothesis, we have correctly determined the cell's contents.
- The cell is in C . We recover the contents from the encoding.
- The cell is on R 's side of the separator between R and W . Then, it was not written during $[i, j-1]$, and thus it has the old value before time i . Given G , everything is fixed before time i , so we know the cell's contents.
- The cell is on W 's side of the separator. Then this thread of execution cannot be in the computation history chosen by the encoding algorithm. We abort the thread.

For each query, there exists a unique partial sum for which it should accept. Furthermore, one accepting thread is included in the computation history of the encoder. Thus, we identify at least one thread which accepts and is not aborted because of the last case above. Because the data structure is correct, all accepting threads must be for the same partial sum, so we correctly identify the sum. By definition of the nondeterministic model, the cell writes are identical for all accepting threads, so we correctly determine the cell writes, as well.

It should be noted that, even though the size of the encoding depends only on the characteristics of one accepting thread per query, the separator allows us to handle an arbitrary number of rejecting threads. All such threads (including all threads for incorrect partial sums) are either simulated until they reject, or they are aborted.

6.4. Handling Monte Carlo randomization. This section shows that the logarithmic lower bound for dynamic connectivity is also true if we allow Monte Carlo randomization with two-sided error probability at most n^{-c} , for constant c (that is, the data structure must be correct with high probability). The idea is to make the decoding algorithm from the previous section use only a polynomial number of calls to data-structure operations.

Assume for now that the data structure is deterministic. The previous decoding algorithm simulates a large number of primitive connectivity operations for every query. A partial sum is recovered by simulating **VERIFY-SUM** for all possible sums. Remember that a partial sum in the dynamic-connectivity problem is a permutation in $S_{\sqrt{n}}$, obtained by composition of the permutations up to a certain column k . Thus, there are $(\sqrt{n})!$ partial sums to try—a huge quantity. However, we can recover the partial-sum permutation by simulating at most $(\sqrt{n})^2$ **CONNECTED** queries: it suffices to test connectivity of every point in the first column with every point on the k th column. First, the decoder simulates **CONNECTED** queries between the first node in column one and every node in column k . Exactly one of these queries was executed by the encoder so that query should accept. The other queries will reject or be aborted. Now the writes made by the accepting query are incorporated in the data structure. The decoder continues to simulate query calls between the second node in column one and all nodes in column k , and so on.

Now assume that the data structure makes an error with probability at most n^{-c} for a sufficiently large constant c . We make the encoding randomized; the random bits are those used to initialize the memory of the data structure. We assume both the encoder and the decoder receive the same random bits, so both can simulate the same behavior of the data structure. By the minimax principle, we can fix those random bits if we are interested only in the expected size of the encoding for a known input distribution (which is the case).

The decoding algorithm described above will work if all correct queries accept, *and* all incorrect queries would reject if they were executed instead of the correct one. We can simulate the execution of any query, or abort it only if it is not one of the correct queries. So if all incorrect queries reject, their simulation will either reject or be aborted. Because we consider only polynomial sequences of operations, we simulate at most $\text{poly}(n)$ queries (including the incorrect ones). The probability that any of them will fail is at most $n^{-c'}$ for some arbitrarily large constant c' (depending on c). Because the decoder has the same coins as the encoder, the encoder can predict whether the decoder will fail. Thus, it can simply add one bit saying whether the old encoding is used (when the decoder works), or the entire input is simply included in the encoding (if the old decoder would fail). The expected size of the encoding grows by at most $1 + n^{-c'} \cdot \text{poly}(n) < 2$ for sufficiently large c' . So the bounds of Lemma 6.1 remain the same. Then, the bounds and trade-offs derived for dynamic connectivity hold even if the data structure answers correctly with high probability.

7. Handling a higher word size. For the partial-sums problem, it is natural and traditional to consider the case $\delta = o(b)$. For ease of notation, we will let $B = \frac{b}{\delta}$. For dynamic connectivity, our motivation comes from external-memory models. For this problem, a “memory cell” is actually an entire page, because that is the unit of memory that can be accessed in constant time. In this case, B is what is usually referred to as “page size”; the number of bits in a page is $b = B \cdot \lg n$. For both problems, the lower bound we obtain is $\Omega(\log_B n)$.

We note that the analysis from the previous sections gives a tight bound on the number of bits that must be communicated: $\Omega(\delta \lg n)$. Given that we can pack b bits into a word, it is straightforward to conclude that $\Omega(\frac{\delta}{b} \lg n) = \Omega(\frac{\lg n}{B})$ read instructions must be performed. Our strategy for achieving $\Omega(\frac{\lg n}{\lg B})$ is to argue that an algorithm cannot make efficient use of all b bits of a word if future queries are sufficiently unpredictable. Intuitively speaking, if we need δ bits of information from a certain time epoch to answer a query, and there are $t \cdot \frac{b}{\delta}$ possible future queries

that would also need δ bits of information from the same epoch ($t > 1$), a cell probe cannot be very effective. No matter what information the cell probe gathers, we have a probability of at most $1/t$ that it has gathered all the information necessary for a random future query, so with constant probability the future query will need another cell probe. The reader will recognize the similarity, at an intuitive level, with the round-elimination lemma from communication complexity [MNSW98, SV]. Also note that our proof strategy hopelessly fails with any deterministic sequence of indices, such as the bit-reversal permutation. Thus, we are identifying another type of hardness hidden in our problems.

Unfortunately, there are two issues that complicate our lower bounds. The first is that, for dynamic connectivity, we need to go beyond the **VERIFY-SUM** abstraction, and deal with **CONNECTED** queries directly. To see why, remember that a **VERIFY-SUM** macro-query accesses a lot of information ($\Theta(\sqrt{n} \lg n)$ bits) in a very predictable fashion, depending on just one query parameter. Thus, we do not have the unpredictability needed by our lower bound. The second complication is that, for the partial-sums problem, we can handle **VERIFY-SUM** only when $\delta = \Omega(b)$. When $\delta = o(\lg n)$, the information per query is not enough to hide the cost of the separators from Lemma 6.2. However, we can still obtain lower bounds for **SUM** and **SELECT**, without nondeterminism, using a rather simple hack.

In section 7.1, we describe a new analysis for adjacent intervals of operations, which is the gist of our new lower bounds. In section 7.2, we show how this new lower bound can be used for the partial-sums problems, whereas in section 7.3, we show how to apply it to dynamic connectivity.

7.1. A new lower bound for adjacent intervals. We now consider an abstract data-structure problem with two operations, **UPDATE** and **QUERY**. We do not specify what **UPDATE** does, except that it receives some parameters and behaves deterministically based on those. A query receives two parameters i and q and returns a Boolean answer. We refer to i as an index. For any admissible i , there exists a unique q which makes the query accept. The parameter q is a δ -bit value, where δ is a parameter of the problem; we let $B = b/\delta$. The implementation of **QUERY** can be nondeterministic. We assume that the hard instance of the problem comes from some random distribution, but that the pattern of updates and queries is deterministic. In the hard instance, each **QUERY** receives the q which makes it accept. We will assume that the random features of each operation are chosen by the distribution independently of the choices for other operations. Though we do not really need this assumption, it is true in both problems we consider and, assuming independence, simplifies exposition.

Now consider two intervals of operations $[i, j - 1]$ and $[j, k]$, and let L be the number of queries in the second interval. We make an information-theoretic assumption, which we will later prove for both the partial-sums and dynamic-connectivity problems. To describe this assumption, pick a random $t \in [j, k]$ such that the t th operation is a query. Also pick a set Q of BL random queries that could have been generated as query t . Now, imagine simulating each such query starting with the state of the data structure at time $t - 1$. Our assumption is essentially that the correct q 's for all original queries from $[j, k]$, plus the simulated queries in Q , have high entropy. More specifically, let Z be the random variable specifying all updates outside $[i, j - 1]$ and the indices for all queries, including those in Q . We assume that the vector of q 's has entropy $\Omega(BL\delta)$ given Z .

Let w be the number of write instructions executed during the first interval, and

let r be the number of read instructions executed during the second interval. Also let c be the number of read instructions executed during the second interval that read cells last written during the first interval. Under the assumptions above, we prove $E[c] = \Omega(L) - O(\frac{s}{b})$, where $s = (B \cdot E[r]) \log_2(1 + \frac{E[w]}{B \cdot E[r]})$.

We now outline the proof strategy. The probability that query t reads at least one cell from $[i, j - 1]$ is at most $\frac{E[c]}{L}$. If $E[c]$ were small, so would this probability. That would mean that a large fraction of the queries from Q (random queries that could be executed at time t) would not need to read any cell written during $[i, j]$. On the other hand, the queries recover $\Omega(BL\delta) = \Omega(Lb)$ bits of information about the updates in the left interval. Because most queries don't need to read another cell, most of this information must have already been recovered by the cell probes made in $[j, t - 1]$. There are at most $E[c]$ probes in expectation, each reading b bits, so the recovered information is not enough when $E[c]$ is small.

Encoding algorithm. As the first step of the formal proof, we describe the algorithm encoding the correct q 's. First, simulate the entire execution of the data structure, with the real query at time t . For each query, include an arbitrary accepting thread in the computation history. Based on this computation history, consider the following sets:

- C = cells that are written during $[i, j - 1]$ and read during $[j, k]$.
- W = cells that are written during $[i, j - 1]$ but not read during $[j, k]$.
- R_1 = cells that are read during $[j, k]$, but never written during $[i, j - 1]$.

Now simulate the queries in Q starting from the state of the data structure at time $t - 1$. As before, we only pass correct parameters to these queries. Call *easy queries* the queries for which there exists an accepting thread which does not read any cell in W ; call such a thread a *good thread*. The rest of the queries are *hard queries*; let h be the number of hard queries. Let R_2 be the union of the cells read by an arbitrary good thread of every easy query, from which we exclude the cells in C . By the definition of easy queries, R_2 is disjoint from W . Let $R = R_1 \cup R_2$; R is also disjoint from W .

The encoding has four parts as follows:

1. Encode c and, for each cell in C , the address and contents of the cell;
2. a separator (as given by Lemma 6.2) between R and W ;
3. encode h , and the set of hard queries. The set takes $\lg \binom{|Q|}{h} = \lg \binom{BL}{h}$ bits;
4. the correct q for each hard query, as an array of size h . This takes $h\delta$ bits.

The third part of the encoding could be avoided for our current problem, because the separator can be used to recognize hard queries. However, we will later consider a variation in which we discard the separator, and then encoding which queries are hard could no longer be avoided.

Decoding algorithm. We now describe how to recover the correct q 's given Z and the previous encoding. By definition, a separator of R and W is also a separator for R_1 and W . Given this separator and complete information about C , we can simulate the real operations in the second interval, as argued in Lemma 6.1, and recover their correct q 's. Now we have to recover the correct parameters for the queries in Q . For hard queries, this is included in the encoding. For each easy query, each possible q , and all threads, we try to simulate the thread starting with what we know about the data structure at time $t - 1$. Each cell that is probed falls into one the following cases:

- The cell was written during $[j, t - 1]$. Because we simulated the data structure in this interval, we can identify this condition and recover the cell contents.
- The cell is in C . We recover the contents from the encoding.

- The cell is on R 's side of the separator between R and W . Then, it was not written during $[i, j - 1]$, and we can recover the cell contents because Z includes perfect information before time i .
- The cell is on W 's side of the separator. Then, this thread of execution cannot be among the chosen good threads for the easy queries, so we abort it.

For the chosen good thread of an easy query, the encoder included its probes outside of C in the set R_2 , so simulation of this thread is never aborted. Thus, for each easy query, we find at least one accepting thread, and recover the correct q .

Analysis. We want to bound the size of the separator. We have $|W| \leq w$, $|R_1| \leq r$, so it remains to bound $|R_2|$. In expectation over a random t and a random choice of the queries in the second interval, the number of cells read by query t is at most $\frac{E[r]}{L}$. We are simulating a set of BL queries as if they happened at time t . In expectation, the total number of cell probes performed by these is at most $BL \frac{E[r]}{L} = B \cdot E[r]$, which also bounds $E[|R_2|]$. Then $E[|R|] \leq E[|R_1|] + E[|R_2|] = O(B)E[r]$. To specify the separator, we need $O(b)$ bits to write $|W|$ and $|R|$, and then, by Lemma 6.2, $O(\lg b + \log_2 \binom{|W|+|R|}{|R|})$ bits for the index into the system of separators. The total size is $O(b + |R| \log_2(1 + \frac{|W|}{|R|}))$ bits. The function $(x, y) \mapsto x \log_2(1 + \frac{y}{x})$ is concave, so the expected size is upper bounded by moving expectations inside. Then, the expected size of the separator is $O(b + (B \cdot E[r]) \lg(1 + \frac{E[w]}{B \cdot E[r]})) = O(b + s)$.

To analyze the rest of the encoding, we need to bound h . For a random t , the expected number of cell probes from the first interval that are made by query t is at most $\frac{E[c]}{L}$. This means that a random query at position t is bad with probability at most $\frac{E[c]}{L}$. Thus, $E[h] = BL \frac{E[c]}{L} = B \cdot E[c]$. Explicitly encoding the correct q 's for the hard queries takes $E[h]\delta = b \cdot E[c]$ bits in expectation. This is the same as the space taken to encode the contents of cells in C . Encoding which queries are hard takes space $O(b) + \lg \binom{BL}{h} = O(b + h \lg \frac{BL}{h})$. The function $x \rightarrow x \lg \frac{\gamma}{x}$ is concave for constant γ , so the expected size is at most $O(b + E[h] \lg \frac{BL}{E[h]}) = O(b + B \cdot E[c] \lg \frac{L}{E[c]})$.

We have shown an upper bound of $O(E[c]b + s + B \cdot E[c] \lg \frac{L}{E[c]})$ on the expected total size of the encoding. Let $\varepsilon > 0$ be an absolute constant to be determined. If $E[c] \geq \varepsilon L$, there is nothing to prove. Otherwise, observe that $x \mapsto x \log_2(2 + \frac{\gamma}{x})$ grows with x for constant γ , so the last term of the encoding size becomes $O(B\varepsilon L \lg \frac{1}{\varepsilon})$. The assumed lower bound on the size of the encoding is $\Omega(BL\delta) = \Omega(bL)$, so we obtain $E[c] = \Omega(L) - O(\frac{\varepsilon}{\delta} L \lg \frac{1}{\varepsilon}) - O(\frac{s}{b})$. Note that $\varepsilon \lg \frac{1}{\varepsilon}$ goes to zero as ε goes to zero. Then, assuming $\delta \geq 2$, there is an absolute constant ε such that the second term of the lower bound is a constant fraction of the first term. We thus obtain $E[c] = \Omega(L) - O(\frac{s}{b})$.

Deterministic queries. Now we consider a variation of our original problem, in which queries are deterministic, and they return q , as opposed to verifying a given q . The only change in our analysis is that we do not need the separator. Indeed, each query can be simulated unambiguously, because it only receives a known index, and it is deterministic. Then, the separator term in our lower bound disappears, and we obtain $E[c] = \Omega(L)$.

7.2. The partial-sums problem. Our hard instance is the same as in section 5.5: we consider blocks of t_q random updates and t_u queries to random indices. We begin by showing a lower bound for two intervals based on the analysis from the previous section. Let c, r, w be as defined in the previous section.

LEMMA 7.1. *Consider two adjacent intervals of operations such that the left interval contains $B \cdot L$ updates, the right interval contains L queries, and overall the intervals contain $O(\sqrt[3]{n})$ operations. The following lower bounds hold:*

- In the case of **SUM** queries, $E[c] = \Omega(L)$.
- In the case of **VERIFY-SUM** queries, $E[c] = \Omega(L) - O(\frac{s}{b})$, where we define $s = (B \cdot E[r]) \log_2(1 + \frac{E[w]}{B \cdot E[r]})$.

Proof. This follows from the analysis in the previous section, as long as we can show the information-theoretic assumption made there. Specifically, we pick a query from the second interval, and imagine simulating BL random queries in its place. We need to show that the partial sums of the original queries and these virtual queries have entropy $\Omega(BL\delta)$, given the indices of all queries, and the indices and Δ values for all queries outside the left interval (the variable Z from the previous section). To prove this, we apply Lemma 5.3. Because that lemma deals only with the partial sums (a feature of the problem instance) and not with computation, it doesn't matter that we are simulating the BL queries at the same time. The partial sums would be the same if the queries ran consecutively. Then, the lemma applies and shows our entropy lower bound. Note that the variable G in Lemma 5.3 describes all queries, including the simulated ones (which the lemma thinks are consecutive). This is exactly the variable Z . \square

We now show how to use this lemma to derive our lower bounds. Our analysis is similar to that of section 6.2, with two small exceptions. The first is that there is an inherent asymmetry between the left and right intervals in Lemma 7.1. Because of this, we can handle only the case $t_q = O(t_u)$. The second change is that the definition of s is somewhat different from that in Lemma 6.1; roughly, s is larger because $E[r]$ is multiplied by B . We will show lower bounds of the form $t_q(\lg B + \lg \frac{t_u}{t_q}) = \Omega(\lg n)$.

We consider a balanced tree with branching factor $\beta = 2B \frac{t_u}{t_q}$ over $m = \Theta(\sqrt[n]{n})$ blocks. Because, for $\max\{t_q, t_u\} = \Omega(\sqrt[n]{n})$, our trade-off states $\min\{t_q, t_u\} = \Omega(1)$, we may assume $t_u + t_q = O(\sqrt[n]{n})$. Then there are $O(\sqrt[n]{n})$ operations in total, as needed. We will consider right intervals formed by a node of the tree, and left intervals formed by all its left siblings. The choice of β gives the right proportion of updates in the left interval, compared to queries in the right interval, for any node which is in the right half of its siblings. Then, we can apply Lemma 7.1.

First, consider the case of **SUM** queries, so there is no term depending on s . Note that the $\Omega(L)$ term is linear in the number of queries, so summing it over the entire tree yields a lower bound on the total time of $E[T] = \Omega(mt_u \log_\beta n)$. By the definition of the blocks, $E[T] = m \cdot 2t_u t_q$, so $t_q = \Omega(\log_\beta n)$, which is equivalent to $t_q(\lg B + \lg \frac{t_u}{t_q}) = \Omega(\lg n)$.

Now we consider nondeterministic **VERIFY-SUM** queries, assuming $\delta = \Omega(\lg n)$. There is a new term in the lower bound on $E[T]$, given by the sum of the s terms over all nodes. First, consider the sum for all nodes on one level: $\sum s_i = \sum (B \cdot E[r_i]) \log_2(1 + \frac{E[w_i]}{B \cdot E[r_i]})$. We have $\sum r_i \leq T$, because each read is counted for the node it is under, and $\sum w_i \leq \beta T$, because each write is counted for the siblings of the node it is under. These inequalities must also hold in expectation, so $\sum B \cdot E[r_i] \leq B \cdot E[T]$ and $\sum E[w_i] \leq \beta E[T]$. Because the function $(x, y) \mapsto x \log_2(1 + \frac{y}{x})$ is concave, $\sum s_i$ is maximized when $E[r_i]$ and $E[w_i]$ are equal. Then $\sum s_i \leq B \cdot E[T] \log_2(1 + \frac{\beta E[T]}{B \cdot E[T]}) = O(E[T] B \lg(t_u/t_q))$. When we sum this over $O(\log_\beta n)$ levels, we obtain $E[T] \cdot O(B \lg(t_u/t_q) \frac{\lg n}{\lg(B t_u/t_q)}) = E[T] \cdot O(B \lg n)$.

Thus, our overall lower bound becomes $E[T] = \Omega(mt_u \log_\beta n) - O(\frac{1}{b} E[T] B \lg n)$. Expanding B , $E[T] = \Omega(mt_u \log_\beta n) - O(E[T] \frac{\lg n}{\delta})$. For $\delta = \Omega(\lg n)$, we obtain $E[T] = \Omega(mt_u \log_\beta n)$, which is the same lower bound as for **SUM** queries. This bound holds for **VERIFY-SUM** queries, even with nondeterminism, and, as shown in section 6.1,

also for **SELECT** queries.

For the case $\delta = O(\lg n)$, we can obtain the same lower bound on **SELECT** by a relatively simple trick. This means that the trade-off lower bound for **SELECT** holds for any δ , though we cannot prove it in general for **VERIFY-SUM**. The trick is to observe that for small δ (e.g., $\delta < \frac{1}{3} \lg n$), we can stretch (polynomially) an instance of **SUM** into an instance of **SELECT**. Because we already have a lower bound for **SUM**, a lower bound for **SELECT** follows by reduction.

Consider the **SUM** problem on an array $A[0.. \sqrt[3]{n} - 1]$, where each element has δ bits. This implies $0 \leq A[i] < \sqrt[3]{n}$, and any partial sum is less than $n^{2/3}$. Now we embed A into an array $A'[0.. n-1]$ by $A'[i \cdot n^{2/3}] = A[i]$. The $n^{2/3} - 1$ spacing positions between elements from A are set to 1 in the initialization phase, and never changed later. An update in A translates into an update in A' in the appropriate position. Now assume we want to find $\sigma = \sum_{i=0}^k A[i]$. We run **SELECT** $((k+1)(n^{2/3} - 1))$ in A' . We have $\sum_{j=0}^{t+k \cdot n^{2/3}} A[j] = t + \sigma + k(n^{2/3} - 1)$, for any $t < n^{2/3}$. Then, if **SELECT** returns $t + k \cdot n^{2/3}$, we know that $t + \sigma = n^{2/3} - 1$, so we find σ .

7.3. Dynamic connectivity. The graph used in the hard sequence is the same as the one before (Figure 6.1): \sqrt{n} permutation boxes, each permuting \sqrt{n} “wires.” Let t_u be the running time of an update (edge insertion or deletion) and t_q the running time of a query. We handle only $t_q \leq t_u$. Our hard sequence consists of blocks of operations. Each block begins with a macro-update: for an index k (chosen as described below), remove all edges in the k th permutation box, and insert edges for a random permutation. Then, the block contains $\frac{t_u}{t_q} \sqrt{n}$ **CONNECTED** queries. Each query picks a random node in the first column and a random index k , and calls **CONNECTED** on the node in the first column and the node on the k th column which is on the same path. This means that all queries should be answered in the affirmative; the information is contained in the choice of the node from the k th column.

We still have to specify the sequence of indices of the macro-updates. We use a deterministic sequence to ensure that updates which occur close in time touch distant indices. This significantly simplifies the information-theoretic analysis. Our hard sequence consists of exactly \sqrt{n} blocks. Each macro-update touches a different permutation box; the order of the boxes is given by the bit-reversal permutation (see section 5.6) of order \sqrt{n} . Now consider a set of indices $S = \{i_1, i_2, \dots\}$ sorted by increasing i_j . We say S is *uniformly spaced* if $i_{j+1} - i_j = \sqrt{n}/(|S| - 1)$ for every j .

LEMMA 7.2. *Consider two adjacent intervals of operations such that the second one contains L queries, and the indices updated in the first interval contain a uniformly spaced subset of cardinality $\Theta(BL/\sqrt{n})$. Then $E[c] = \Omega(L) - O(\frac{s}{b})$, where $s = B \cdot E[r] \lg(1 + \frac{E[w]}{B \cdot E[r]})$.*

Proof. This lemma follows from section 7.1, if we show the information-theoretic assumption used there. For our problem, $\delta = \Theta(\lg n)$. Imagine picking a random query from the right interval and simulating BL random queries in its place. The variable Z denotes the random choices for all queries and for updates outside the left interval. We need to show that the entropy of the correct parameters for all queries in the right interval, including the simulated ones, given Z , is $\Omega(BL \lg n)$.

Remember that all updates are to different boxes, so an update is never overwritten. For this reason, our proof is not concerned with the precise order of updates and queries in the right interval, and there will be no difference between the real and simulated queries. Let the uniformly spaced set of update indices be $S = \{i_1, i_2, \dots\}$. We let B_j be the set of queries from the right interval (either real or simulated) whose

random column is in $[i_j, i_{j+1} - 1]$. For notational convenience, we write $H(B_j)$ for the entropy of the correct parameters to the set of queries B_j . Basic information theory states that $H(\bigcup_j B_j \mid Z) = \sum_j H(B_j \mid B_1, \dots, B_{j-1}, Z)$. Thus, to prove our lower bound, it suffices to show $H(B_j \mid B_1, \dots, B_{j-1}, Z) = \Omega(\sqrt{n} \lg n)$ for all j .

Let Z_j be a random variable describing Z and, in addition, the random permutations for all updates in the left interval with indices below i_j . Also, if there are any updates with indices in $[i_j + 1, i_{j+1} - 1]$, include their permutation in Z_j (these are updates outside the uniformly spaced set). Note that $H(B_j \mid B_1, \dots, B_{j-1}, Z) \geq H(B_j \mid Z_j)$, because conditioning on Z_j also fixes the correct parameters for queries in B_1, \dots, B_{j-1} .

Now let us look at a query from B_j . The query picks a random node in the first column. All permutations before column i_j are fixed through Z_j , so we can trace the path of the random node until it enters box i_j . Assume we have the correct parameter of the query, i.e., the node from column k to which the initial node is connected. Permutations between column i_j and i_{j+1} are also fixed by Z_j , so we can trace back this node until the exit of box i_j . Thus, knowing the correct parameter is equivalent to knowing some point values of the permutation i_j . As long as the nodes chosen in the first column are distinct, we will learn new point values. If we query d distinct point values of the random permutation, the entropy of the correct parameters is $\Omega(d \lg n)$, for any d .

Now imagine an experiment choosing the queries sequentially. This describes a random walk for d . In each step, d may remain constant or it may be incremented. Because of the uniform spacing, the probability that a query ends up in B_j is $\Omega(\frac{\sqrt{n}}{BL})$. If $d \leq \sqrt{n}/2$, with probability at least half, the node chosen in the first column is new. Then, for $d \leq \sqrt{n}/2$, the probability that d is incremented is $\Omega(\frac{\sqrt{n}}{BL})$. We do BL independent random steps, and we are interested in the expected value of d at the end. The waiting time until d is incremented is $O(\frac{BL}{\sqrt{n}})$. For a sufficiently small constant ε , the expected time until d reaches $\varepsilon\sqrt{n}$ is $\frac{1}{2}BL$. Then, with probability at least half, $d \geq \varepsilon\sqrt{n}$ after BL steps. This implies the expected value of d after BL steps is $\Omega(\sqrt{n})$, so $H(B_j \mid Z_j) = \Omega(\sqrt{n} \lg n)$. \square

To use this lemma, we construct a tree with a branching factor $\beta \geq 2B \frac{t_u}{t_q}$, rounded to the next power of 2. The right interval is formed by a node, and the left interval by the node's left siblings. We consider only the case when the node is among the right half of its siblings. Now we argue there is a uniformly spaced subset among the indices updated in the left interval. Note that these include all indices from the first half of siblings. Because β is a power of 2, a root-to-leaf path in the tree is tracing a bit representation of the leaf's index, in chunks of $\log_2 \beta$ bits. Because update indices are the reverse of the leaf's index, all the leaves in the subtrees of the first half of the children have the same low order bits in the indices. On the other hand, the high order bits assume all possible values. So the indices from the first half of the children are always a uniformly spaced subset of indices.

Now we can apply Lemma 7.2, and we sum over all nodes of the tree to obtain our lower bound. By the analysis in the previous section, the sum of the s terms only changes the bound by a constant factor. The $\Omega(L)$ term of the lower bound is linear in the number of queries, so by summing over all levels we obtain $E[T] = \Omega(\sqrt{n} \cdot \frac{t_u}{t_q} \sqrt{n} \cdot \log_\beta n)$. Because $E[T] = \sqrt{n}(t_u \sqrt{n} + t_q \frac{t_u}{t_q} \sqrt{n}) = O(\sqrt{n} \cdot t_u \sqrt{n})$, we obtain $t_q = \Omega(\log_\beta n)$, which is our desired lower bound.

8. Upper bounds for the partial-sums problem. As mentioned before, our partial-sums data structure can support a harder variant of updates. We will allow the $A[i]$'s to be arbitrary b -bit integers, while $\text{UPDATE}(i, \Delta)$ implements the operation $A[i] \leftarrow A[i] + \Delta$, where Δ is a δ -bit (signed) integer.

Our data structure is based on a balanced tree of branching factor B (to be determined) with the elements of the array $A[1..n]$ in the leaves. Assume we pick B such that we can support constant-time operations for the partial-sums problem in an array of size B . Then, we can hold an array of size B in every node, where each element is the total of the leaves in one of the B subtrees of our node. All three operations in the large data structure translate into a sequence of operations on the small data structures of the nodes along a root-to-leaf path. Thus, the running time is $O(\log_B n)$. We will show how to handle $B = \Theta(\min\{b/\delta, b^{1/5}\})$. Then $\lg B = \Theta(\lg(b/\delta))$, which implies our upper bound.

It remains to describe the basic building block, i.e., a constant-time solution for arrays of B elements. We now give a simple solution for UPDATE and SUM . In the next section, we develop the ideas necessary to support SELECT . We will conceptually maintain an array of partial sums $S[1..B]$, where $S[k] = \sum_{i=1}^k A[i]$. To make it possible to support UPDATE in constant time, we maintain the array as two separate components, $V[1..B]$ and $T[1..B]$, such that $S[i] = V[i] + T[i]$. The array V will hold values of S that were valid at some point in the past, while more recent updates are reflected only in T . We can use Dietz's incremental rebuilding scheme [Die89] to keep every element of B relatively up to date: on the t th UPDATE , we set $V[t \bmod B] \leftarrow V[t \bmod B] + T[t \bmod B]$ and $T[t \bmod B] \leftarrow 0$. This scheme guarantees that every element in T is affected by at most B updates, and thus is bounded in absolute value by $B \cdot 2^\delta$.

The key idea is to pack T in a machine word. We represent each $T[i]$ by a range of $O(\delta + \lg n)$ bits from the word, with one zero bit of padding between elements. Elements in T also can be negative; in this case, each value will be represented in the standard two's complement form on its corresponding range of bits. Packing T in a word is possible as long as $B = O(\frac{b}{\delta + \lg b})$. We can read and write an element of T using a constant number of standard RAM operations (bitwise Boolean operations and shift operations).

To complete our solution, we need to implement UPDATE in constant time. Using the packed representation, we can add a given value to all elements $V[i]$, $i \geq k$, in constant time. Refer to Table 8.1. First, we create a word with the value to be added appearing in all positions corresponding to the elements of V that need to be changed. We can compute this word using a multiplication by an appropriate binary pattern. The result is then added to the packed representation of V ; all the needed additions are performed in one step, using word-level parallelism. Because we are representing negative quantities in two's complement, additions may carry over, and set the padding bits between elements; we therefore force these buffer bits to zero using a bitwise AND with an appropriate constant mask.

8.1. Selecting in small arrays. To support SELECT , we use the classic result of Fredman and Willard [FW93] that forms the basis of their fusion-tree data structure. Their result has the following black-box functionality: for $B = O(b^{1/5})$, we can construct a data structure that can answer successor queries on a static array of B integers in constant time. As demonstrated in [AMT99], the lookup tables used by the original data structure can be eliminated if we perform a second query in the sketch representation of the array. The data structure then can be constructed in $O(B^4)$ time.

TABLE 8.1
Performing UPDATE(2, Δ) at the word level. Here V has 5 elements, each 5 bits long.

V[4] 0 V[3] 0 V[2] 0 V[1] 0 V[0]	old packed representation of V
00001 0 00001 0 00001 0 00001 0 00001	constant pattern
00001 0 00001 0 00001 0 00000 0 00000	shift right, then left by same amount
Δ	argument given to UPDATE
Δ 0 Δ 0 Δ 0 00000 0 00000	multiply the last two values
V'[4] ? V'[3] ? V'[2] ? V'[1] ? V'[0]	add to the packed representation of V
11111 0 11111 0 11111 0 11111 0 11111	constant cleaning pattern
V'[4] 0 V'[3] 0 V'[2] 0 V'[1] 0 V'[0]	final value of V, obtained by bitwise AND

As before, we break partial sums into the arrays V and T . We store a fusion structure that can answer successor queries in V . Because the fusion structure is static, we abandon the incremental rebuilding of V in favor of periodic global rebuilding. By the standard deamortization of global rebuilding [dBSvKO00], we can then obtain worst-case bounds. Our strategy is to rebuild the data structure completely every B^4 operations: we set $V[i] \leftarrow V[i] + T[i]$ and $T[i] \leftarrow 0$, for all i , and rebuild the fusion structure over V . While servicing a **SELECT** that doesn't occur immediately after a rebuild, the successor in V found by the fusion structure might not be the appropriate answer to the **SELECT** query, because of recent updates. We will describe shortly how the correct answer can be computed by also examining the array T ; the key realization is that the real successor must be close to the successor in V in terms of their partial sums.

Central to our solution is the way we rebuild the data structure every n^4 operations. We begin by splitting S into runs of elements satisfying $S[i + 1] - S[i] < B^4 \cdot 2^\delta$; recall that we must have $S[i] < S[i + 1]$ for the **SELECT** problem. We denote by $\text{rep}(i)$ the first element of the run containing i (the representative of the run); also let $\text{len}(i)$ be the length of the run containing i . Each of these arrays can be packed in a word, because we already limited ourselves to $B = O(b^{1/5})$. Finally, we let every $V[i] \leftarrow V[\text{rep}(i)]$ and $T[i] \leftarrow S[i] - V[\text{rep}(i)]$. Conveniently, T can still be packed in a word. Indeed, the value stored in an element after a rebuild is at most $B \cdot (B^4 \cdot 2^\delta)$, and it can subsequently change by less than $B^4 \cdot 2^\delta$. Therefore, it takes $O(\lg B + \delta)$ bits to represent an element of C , so we need only impose the condition that $B = O(\min\{b/\delta, b^{1/5}\})$.

It remains to show how **SELECT**(σ) can be answered. Let k denote the successor in V identified by the fusion structure; we have $V[k - 1] < \sigma \leq V[k]$. We know that k is the representative of a run, because all elements of a run have equal values in V . By construction, runs are separated by gaps of at least $B^4 \cdot 2^\delta$, which cannot be closed by B^4 updates. Thus, the answer to the query must be either an index in the run starting at k , or an index in the run ending at $k - 1$, or exactly equal to $k + \text{len}(k)$. We can distinguish between these cases in constant time, using two calls to **SUM** followed by comparisons. If we identify the correct answer as exactly $k + \text{len}(k)$, we are done.

Otherwise, assume by symmetry that the answer is an index in the run starting at k . Because elements of a run have equal values of V , our task is to identify the unique index i in the run satisfying $T[i - 1] < \sigma - V[k] \leq T[i]$. Now we can employ word-level parallelism to compare all elements in T with $\sigma - V[k]$ in constant time. This is similar to a problem discussed by Fredman and Willard [FW93], but we must also handle negative quantities. The solution is to subtract $\sigma - V[k]$ in parallel from all elements in T ; if elements of T are oversized by 1 bit, we can avoid overflow. The

sign bits of every element then give the results of the comparisons. The answer to the query can be found by summing up the sign bits corresponding to elements in our run, which indicates how many elements in the run were smaller than $\sigma - V[k]$. Because these bits are separated by more than $\lg b$ zeros, we can sum them up using a multiplication with a constant pattern, as described in [FW93].

9. Reductions to other dynamic graph problems. It is relatively easy to dress up dynamic connectivity as other dynamic graph problems, obtaining logarithmic lower bounds for these. Most problems on undirected graphs admit polylogarithmic solutions, so such lower bounds are interesting. The problems discussed in this section are meant only as examples and not as an exhaustive list of possible reductions.

9.1. Connectivity of the entire graph. The problem is to maintain a dynamic graph along with the answer to the question, “Is the entire graph connected?”. We obtain a lower bound of $\Omega(\lg n)$ even for plane graphs, which implies the same lower bound for counting connected components. The dynamic-connectivity algorithms mentioned in the introduction also can maintain the number of connected components, so the same almost-tight upper bounds hold for this problem.

We use the same graph as in the dynamic-connectivity lower bound, except that we add a new vertex s which is connected to all nodes from the first column. The updates in the connectivity problem translate into identical updates in our current problem. The hard instance of connectivity asks queries between a vertex u on the first column, and an arbitrary vertex v . To simulate these, we disconnect u from s , connect v to s , and ask whether the entire graph is connected; after this, we undo the two changes to the graph. If u and v were on distinct paths, u 's path will now be disconnected from the rest of the graph. Otherwise, the edge (v, s) will reconnect the path to the rest of the graph.

The graph we consider is a tree, so it is plane regardless of the embedding of the vertices. During a query, if u and v are on the same path, we create an ephemeral cycle. However, the (v, s) edge can simply be routed along the old path $s \rightarrow u \rightsquigarrow v$, so the graph remains plane.

9.2. Dynamic MSF. The problem is to maintain the cost of the MSF (minimum spanning forest) in a weighted dynamic graph. The problem can be solved in $O(\lg^4 n)$ time per operation [HdLT01]. In plane graphs, the problem admits a solution in time $O(\lg n)$ [EIT⁺92]. We obtain a lower bound of $\Omega(\lg n)$, which holds even for plane graphs with unit weights. Our bound follows immediately from the previous bound. If all edges have unit weight and the graph is connected, the weight of the MSF is $n - 1$. If the graph is disconnected, the weight of the MSF is strictly smaller.

9.3. Dynamic planarity testing. The problem is to maintain a dynamic plane graph, and test whether inserting a given edge would destroy planarity. Actual insertions always maintain planarity; an edge (u, v) is given along with an order inside the set of edges adjacent to u and v . The problem can be solved in $O(\lg^2 n)$ time per operation [IPH93]. A lower bound of $\Omega(\lg n / \lg \lg n)$ appears in [FH98]. We obtain a lower bound of $\Omega(\lg n)$.

Because the graph from our lower bound proof is always a collection of disjoint paths, it is plane under any embedding. Consider the complete bipartite graph $K_{3,3}$, from which an edge (s, t) is removed. Without that edge, this annex graph is also planar. To implement connectivity queries between two nodes u and v , we first insert the edge (u, s) temporarily and then query whether inserting the edge (v, t) would

destroy planarity. If u and v are on distinct paths, the graph created by adding (u, s) and (v, t) is planar and can be embedded for any relative order of these two edges (the edges of $K_{3,3} \setminus \{(s, t)\}$ can simply go around the two paths containing u and v). If u and v are on the same path, we would be creating a subdivision (graph expansion) of $K_{3,3}$, so the graph would no longer be planar (by Kuratowski's theorem).

10. Open problems. This paper provides powerful techniques for understanding problems which have complexity around $\Theta(\lg n)$. The chronogram technique had already proven effective for problems with complexity $\Theta(\frac{\lg n}{\lg \lg n})$. However, current techniques seem ineffective either below or above these thresholds. Below this regime, we have integer search problems, such as priority queues. Looking at higher complexities, we find many important problems which seem to have polylogarithmic complexity (such as range queries in higher dimensions) or even $n^{\Omega(1)}$ complexity (dynamic problems on directed graphs). It is also an important complexity theoretic challenge to obtain an $\omega(\lg n)$ lower bound for a dynamic language membership problem.

It is also worth noting that our bounds do not bring a complete understanding of the partial-sums problem when $\delta = o(b)$. First, we cannot prove a tight bound for **VERIFY-SUM**. A bound of $\Omega(\lg n / \lg b)$, for any δ , is implicit in [HR03] and also can be reproved using our technique. Second, we do not have a good understanding of the possible trade-offs. For **SELECT**, this seems a thorny issue, because of the interaction with the predecessor problem. Even for **SUM**, we do not know what bounds are possible in the range $t_u < t_q$. It is tempting to think that the right bound is $t_u(\lg \frac{t_q}{t_u} + \lg \frac{b}{\delta}) = \Theta(\lg n)$, by symmetry with the case $t_u > t_q$. However, buffer trees [Arg03] give better bounds for some choices of parameters, e.g., when $b = \Omega(\lg^{1+\varepsilon} n)$. This problem seems to touch on a fundamental issue: a good lower bound apparently needs to argue that the data structure has to recover a lot of information about the indices of updates, in addition to the Δ values.

It would be very interesting to obtain a logarithmic upper bound for dynamic connectivity matching our lower bound. It also would be interesting to determine the complexity of decremental connectivity. For this problem, at least our trade-off lower bound cannot hold, because [HK99] gave a solution with polylogarithmic updates and constant query time.

REFERENCES

- [AHR98] S. ALSTRUP, T. HUSFELDT, AND T. RAUHE, *Marked ancestor problems*, in Proceedings of the 39th Annual IEEE Symposium on Foundations of Computer Science (FOCS), 1998, pp. 534–543.
- [Ajt88] M. AJTAI, *A lower bound for finding predecessors in Yao's cell probe model*, *Combinatorica*, 8 (1988), pp. 235–247.
- [AMT99] A. ANDERSSON, P. B. MILTERSEN, AND M. THORUP, *Fusion trees can be implemented with AC^0 instructions only*, *Theoret. Comput. Sci.*, 215 (1999), pp. 337–344.
- [Arg03] L. ARGE, *The buffer tree: A technique for designing batched external data structures*, *Algorithmica*, 37 (2003), pp. 1–24. See also WADS '95.
- [BAG01] A. M. BEN-AMRAM AND Z. GALIL, *A generalization of a lower bound technique due to Fredman and Saks*, *Algorithmica*, 30 (2001), pp. 34–66. See also FOCS '91.
- [BAG02] A. M. BEN-AMRAM AND Z. GALIL, *Lower bounds for dynamic data structures on algebraic RAMs*, *Algorithmica*, 32 (2002), pp. 364–395. See also FOCS '91.
- [BF02] P. BEAME AND F. E. FICH, *Optimal bounds for the predecessor problem and related problems*, *J. Comput. System Sci.*, 65 (2002), pp. 38–72. See also STOC '99.
- [CFG+78] L. CARTER, R. FLOYD, J. GILL, G. MARKOWSKY, AND M. WEGMAN, *Exact and approximate membership testers*, in Proceedings of the 10th Annual ACM Symposium on Theory of Computing (STOC), 1978, pp. 59–65.
- [Cha97] B. CHAZELLE, *Lower bounds for off-line range searching*, *Discrete Comput. Geom.*, 17 (1997), pp. 53–65. See also STOC '95.

- [CT91] T. M. COVER AND J. A. THOMAS, *Elements of Information Theory*, John Wiley & Sons, New York, 1991.
- [dBSvKO00] M. DE BERG, O. SCHWARZKOPF, M. VAN KREVELD, AND M. OVERMARS, *Computational Geometry: Algorithms and Applications*, 2nd ed., Springer-Verlag, Berlin, New York, 2000.
- [Die89] P. F. DIETZ, *Optimal algorithms for list indexing and subset rank*, in Proceedings of the 1st Annual Workshop on Algorithms and Data Structures (WADS), Lecture Notes in Comput. Sci. 382, Springer-Verlag, Berlin, 1989, pp. 39–46.
- [EIT+92] D. EPPSTEIN, G. F. ITALIANO, R. TAMASSIA, R. E. TARJAN, J. R. WESTBROOK, AND M. YUNG, *Maintenance of a minimum spanning forest in a dynamic planar graph*, J. Algorithms, 13 (1992), pp. 33–54. See also SODA '90.
- [Fen94] P. M. FENWICK, *A new data structure for cumulative frequency tables*, Software: Practice and Experience, 24 (1994), pp. 327–336.
- [FH98] M. L. FREDMAN AND M. R. HENZINGER, *Lower bounds for fully dynamic connectivity problems in graphs*, Algorithmica, 22 (1998), pp. 351–362.
- [Fre81] M. L. FREDMAN, *A lower bound on the complexity of orthogonal range queries*, J. ACM, 28 (1981), pp. 696–705.
- [Fre82] M. L. FREDMAN, *The complexity of maintaining an array and computing its partial sums*, J. ACM, 29 (1982), pp. 250–260.
- [FS89] M. FREDMAN AND M. SAKS, *The cell probe complexity of dynamic data structures*, in Proceedings of the 21st Annual ACM Symposium on Theory of Computing (STOC), 1989, pp. 345–354.
- [FW93] M. L. FREDMAN AND D. E. WILLARD, *Surpassing the information theoretic bound with fusion trees*, J. Comput. System Sci., 47 (1993), pp. 424–436. See also STOC '90.
- [HdLT01] J. HOLM, K. DE LICHTENBERG, AND M. THORUP, *Poly-logarithmic deterministic fully-dynamic algorithms for connectivity, minimum spanning tree, 2-edge, and biconnectivity*, J. ACM, 48 (2001), pp. 723–760. See also STOC '98.
- [HF98] H. HAMPAPURAM AND M. L. FREDMAN, *Optimal biweighted binary trees and the complexity of maintaining partial sums*, SIAM J. Comput., 28 (1998), pp. 1–9. See also FOCS '93.
- [HK99] M. R. HENZINGER AND V. KING, *Randomized fully dynamic graph algorithms with polylogarithmic time per operation*, J. ACM, 46 (1999), pp. 502–516. See also STOC '95.
- [HR03] T. HUSFELDT AND T. RAUHE, *New lower bound techniques for dynamic partial sums and related problems*, SIAM J. Comput., 32 (2003), pp. 736–753. See also ICALP '98.
- [HRS96] T. HUSFELDT, T. RAUHE, AND S. SKYUM, *Lower bounds for dynamic transitive closure, planar point location, and parentheses matching*, Nordic J. Comput., 3 (1996), pp. 323–336.
- [HSS03] W.-K. HON, K. SADAKANE, AND W.-K. SUNG, *Succinct data structures for searchable partial sums*, in Proceedings of the 14th Annual International Symposium on Algorithms and Computation (ISAAC), Lecture Notes in Comput. Sci. 2906, Springer-Verlag, Berlin, 2003, pp. 505–516.
- [HT97] M. R. HENZINGER AND M. THORUP, *Sampling to provide or to bound: With applications to fully dynamic graph algorithms*, Random Structures Algorithms, 11 (1997), pp. 369–379. See also ICALP '96.
- [IPH93] G. F. ITALIANO, J. A. LA POUTRÉ, AND M. R. HENZINGER, *Fully dynamic planarity testing in embedded graphs*, in Proceedings of the 1st Annual European Symposium on Algorithms (ESA), Lecture Notes in Comput. Sci. 726, Springer-Verlag, Berlin, 1993, pp. 212–223.
- [Mil99] P. B. MILTERSEN, *Cell probe complexity—a survey*, presented at the Pre-Conference Workshop on Advances in Data Structures at the 19th Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS), 1999. Available online at <http://www.daimi.au.dk/~bromille/Papers/>
- [MNSW98] P. B. MILTERSEN, N. NISAN, S. SAFRA, AND A. WIGDERSON, *On data structures and asymmetric communication complexity*, J. Comput. System Sci., 57 (1998), pp. 37–49. See also STOC '95.
- [MSVT94] P. B. MILTERSEN, S. SUBRAMANIAN, J. S. VITTER, AND R. TAMASSIA, *Complexity models for incremental computation*, Theoret. Comput. Sci., 130 (1994), pp. 203–236. See also STACS '93.

- [RRR01] R. RAMAN, V. RAMAN, AND S. S. RAO, *Succinct dynamic data structures*, in Proceedings of the 7th Annual Workshop on Algorithms and Data Structures (WADS), Lecture Notes in Comput. Sci. 2125, Springer-Verlag, Berlin, 2001, pp. 426–437.
- [ST83] D. D. SLEATOR AND R. E. TARJAN, *A data structure for dynamic trees*, J. Comput. System Sci., 26 (1983), pp. 362–391. See also STOC '81.
- [SV] P. SEN AND S. VENKATESH, *Lower Bounds for Predecessor Searching in the Cell Probe Model*, <http://www.arxiv.org/abs/cs.CC/0309033> (2003). See also ICALP '01 and CCC '03.
- [Tho00] M. THORUP, *Near-optimal fully-dynamic graph connectivity*, in Proceedings of the 32nd Annual ACM Symposium on Theory of Computing (STOC), 2000, pp. 343–350.
- [Wil89] R. WILBER, *Lower bounds for accessing binary search trees with rotations*, SIAM J. Comput., 18 (1989), pp. 56–67. See also FOCS '86.
- [Yao77] A. C.-C. YAO, *Probabilistic computations: Toward a unified measure of complexity*, in Proceedings of the 18th Annual IEEE Symposium on Foundations of Computer Science (FOCS), 1977, pp. 222–227.
- [Yao85] A. C.-C. YAO, *On the complexity of maintaining partial sums*, SIAM J. Comput., 14 (1985), pp. 277–288.

ON SUMS OF INDEPENDENT RANDOM VARIABLES WITH UNBOUNDED VARIANCE AND ESTIMATING THE AVERAGE DEGREE IN A GRAPH*

URIEL FEIGE†

Abstract. We prove the following inequality: for every positive integer n and every collection X_1, \dots, X_n of nonnegative independent random variables, each with expectation 1, the probability that their sum remains below $n+1$ is at least $\alpha > 0$. Our proof produces a value of $\alpha = 1/13 \simeq 0.077$, but we conjecture that the inequality also holds with $\alpha = 1/e \simeq 0.368$.

As an example for the use of the new inequality, we consider the problem of estimating the average degree of a graph by querying the degrees of some of its vertices. We show the following threshold behavior: approximation factors above 2 require far fewer queries than approximation factors below 2. The new inequality is used in order to get tight (up to multiplicative constant factors) relations between the number of queries and the quality of the approximation. We show how the degree approximation algorithm can be used in order to quickly find those edges in a network that belong to many shortest paths.

Key words. Markov inequality, shortest paths

AMS subject classifications. 60E15, 68W20, 68W25

DOI. 10.1137/S0097539704447304

1. A new inequality. For a random variable X , its typical value may be very different from its mean. In particular, the probability that X exceeds its mean may be arbitrarily close to 1. In some special cases (e.g., when X is symmetric around its mean), the probability that X exceeds its mean is at most $1/2$. The purpose of this manuscript is to investigate the probability that X exceeds its mean when X is the sum of n independent random variables. We show that for nonnegative random variables, this probability is bounded away from 1, provided that we give ourselves a little slack in exceeding the mean.

THEOREM 1. *Let X_1, \dots, X_n be arbitrary nonnegative independent random variables, with expectations μ_1, \dots, μ_n , respectively, where $\mu_i \leq 1$ for every i . Let $X = \sum_{i=1}^n X_i$, and let μ denote the expectation of X (hence, $\mu = \sum_{i=1}^n \mu_i$). Then for every $\delta > 0$,*

$$(1) \quad \Pr[X < \mu + \delta] \geq \min[\delta/(1 + \delta), 1/13].$$

The term $\delta/(1+\delta)$ in Theorem 1 is best possible, as X_1 might be a random variable that has value $1 + \delta$ with probability $1/(1 + \delta)$ and value 0 with probability $\delta/(1 + \delta)$, and every other X_i might be a random variable whose value is 1 with probability 1. This gives $\mu_i = 1$ for every i . For this case, $\Pr[X < \mu + \delta] = \Pr[X_1 = 0] = \delta/(1 + \delta)$. For large δ (e.g., $\delta = 1$), it is not true that $\Pr[X \leq \mu + \delta] \geq \delta/(1 + \delta)$. One can take for every i , $X_i = n + \delta$, with probability $1/(n + \delta)$ and 0 otherwise. This gives $\mu_i = 1$ for every i , implying $\mu = n$. For this case, $\Pr[X < n + \delta] = (1 - 1/(n + \delta))^n$, which is roughly $1/e$ for large n .

*Received by the editors November 11, 2004; accepted for publication (in revised form) September 9, 2005; published electronically February 21, 2006.

<http://www.siam.org/journals/sicomp/35-4/44730.html>

†Department of Computer Science and Applied Mathematics, The Weizmann Institute, Rehovot 76100, Israel, and Microsoft Research, One Microsoft Way, Redmond, WA 98052-6399 (uriel.feige@weizmann.ac.il, urifeige@microsoft.com).

Based on the two examples above we make the following conjecture.

CONJECTURE 2. *In the setting of Theorem 1, for every value of δ and n , one of the two examples above is the worst case for $\Pr[X < \mu + \delta]$.*

Conjecture 2, if true, would allow us to replace the constant $1/13$ with $1/e$ in Theorem 1. A conjecture very similar in nature to Conjecture 2 is the following conjecture that was suggested by Samuels in [6].

CONJECTURE 3. *Let X_1, \dots, X_n be arbitrary nonnegative independent random variables, with expectations μ_1, \dots, μ_n , respectively, where $\mu_1 \geq \dots \geq \mu_n$. Then for every $\lambda > \sum_{j=1}^n \mu_j$ there is some $1 \leq i \leq n$ such that $\Pr[\sum_{j=1}^n X_j < \lambda]$ is minimized when the random variables X_j are distributed as follows:*

- For $j > i$, $X_j = \mu_j$ with probability 1.
- For $j \leq i$, $X_j = \lambda - \sum_{k=i+1}^n \mu_k$ with probability $\frac{\mu_j}{\lambda - \sum_{k=i+1}^n \mu_k}$, and $X_j = 0$ otherwise.

The difference between the settings of the two conjectures is that in Conjecture 3 all means are given, whereas in Conjecture 2 only an upper bound on the means is given. The difference in the conclusions of the conjectures is that in Conjecture 3 we may have an arbitrary $i \in \{1, \dots, n\}$, whereas Conjecture 2 effectively states that i is restricted to one of two values, $i \in \{1, n\}$.

Samuels (see [6, 7] and the references therein) proves Conjecture 3 when n (the number of random variables) is at most 4. The case $n = 2$ was proved earlier in [2]. When $n \geq 5$, Samuels shows that Conjecture 3 is true when $\lambda \geq (n - 1) \sum_{j=1}^n \mu_j$. In contrast, in the current paper we are interested in the case (that when put in the framework of Conjecture 3 corresponds to) $\lambda = \delta + \sum_{j=1}^n \mu_j$ with δ fairly small (e.g., $\delta = \mu_1$).

It may be instructive to consider how some standard probabilistic tools relate to Theorem 1. Consider the case that the X_i are identically distributed. Then the central limit theorem implies that when n is large enough, X approaches the normal distribution, and hence $\Pr[X < \mu]$ approaches $1/2$. However, in Theorem 1 the variables X_i may depend on n , and hence n cannot be thought of as being “large enough” with respect to the X_i (even if they are independently and identically distributed (i.i.d.)). This relates to the fact that we place no bounds on the variance of the X_i , and hence standard bounds on deviations of random variables from their expectation (such as Chebyshev’s bound or Chernoff’s bound) are not applicable. The only restriction on the random variables (other than being independent) is their nonnegativity. In particular, this means that X is nonnegative and that Markov’s inequality can be used to show that $\Pr[X \leq \mu + \delta] \geq \delta/(\mu + \delta)$. For the sum of i.i.d. random variables, this bound tends to 0 as n grows (unlike the bound in Theorem 1).

In addition to the work by Samuels mentioned above, the author is aware of some other work of nature similar to Theorem 1. There are results surveyed and developed by Siegel [8] that show that under certain conditions the median of the sum of random variables does not exceed the mean. This holds, for example, for the sum of Bernoulli random variables (if the mean is an integer). The book by Dubins and Savage [4] analyzes strategies for gambling when the goal is to maximize the probability of ending up with a net profit of δ . There the strategies are adaptive (the next gamble may depend on outcomes of previous gambles), and the gambler may quit once a net profit of δ is achieved. One of the main findings of [4] is a set of sufficient conditions under which the strategy of “playing boldly” is optimal. Informally, this strategy tries to reach a net profit of δ (also taking into account previous losses) in one gamble. A simple example is the repeated doubling approach to gain one dollar when

there are 50/50 odds, in which the gambler first gambles one dollar and then doubles the gamble until the first win (or until he/she runs out of money). The scenario in Theorem 1 can be viewed as a version of “how to gamble in parallel,” in which n unbiased gambles with independent outcomes can be placed in parallel in an attempt to reach a net profit of δ units, where each gamble is allowed to risk at most one unit. Our results show that when δ is small (specifically, $\delta \leq 1/12$), having $n > 1$ does not lead to higher probability of achieving a net profit of δ compared to the case that $n = 1$. For larger values of δ , there is an advantage to having $n > 1$, but regardless of the value of n , the probability of achieving a net profit of δ is bounded away from 1. Regardless of the value of δ , it appears to us (though our proof does not actually show this when $\delta > 1/12$) that similar to the “play boldly” principle, the optimal strategy is based on hoping for one successful gamble. (Namely, when δ is small, only one gamble is nonzero, and when δ is large, all gambles are identical, and it suffices for one successful gamble to both reach a profit of δ and cover for all the losses in the other gambles.) Despite similarities in the nature of the results, the proof techniques from [4] and [8] do not appear to be applicable to the setting of Theorem 1.

Theorem 1 can in principle be used whenever one is interested in bounding the probability that the sum of independent random variables significantly exceeds its expectation. However, in many contexts the random variables are known to have some additional properties (e.g., bounded variance), and useful results can also be derived by other means. The application that motivated the development of the inequality (1) is described in section 2.

2. Estimating the average degree. Let $G(V, E)$ be a graph with n vertices. A *degree query* specifies a vertex $v \in V$ and gets in reply d_v , the degree of v in G . We are interested in estimating $m = |E|$ by making only degree queries. Equivalently, we wish to estimate the average degree $d = 2m/n$. We say that an algorithm provides a ρ estimation if its output d^* satisfies

$$d^* \leq d \leq \rho d^*.$$

Naturally, we limit our interest to $\rho \geq 1$. As our sampling-based algorithms are randomized, there is some probability that their output fails to be a ρ estimation. We require this failure probability to be at most $1/3$. We note that the failure probability can be reduced to an arbitrarily small value δ by repeating the estimation algorithm $O(\log 1/\delta)$ times independently and outputting the median of all estimates. Our goal is for given ρ to design ρ estimation algorithms with as few queries as possible and with failure probability at most $1/3$.

Let us note here an observation that helps us to drastically reduce the number of queries in our algorithms. Consider first the case where rather than having an actual graph as input, the input is simply a sequence of integers d_1, \dots, d_n , with the only restriction that for every i , $0 \leq d_i \leq n$. (For simplicity of presentation we allow values here to range up to n , even though degrees can range only up to $n - 1$.) Let $d = \frac{1}{n} \sum_{i=1}^n d_i$. We wish to estimate d . It is not hard to see that for any value d_0 (which one may think of as a large constant independent of n), $\Omega(n/d_0)$ queries are required in order to distinguish between the cases $d = 0$ and $d \geq d_0$. The reason is that it may happen that there are d_0 numbers with value n , and all other numbers have value 0. If we perform less than $n/2d_0$ queries, most likely we always get the 0 answer, which is exactly the answer that we would get if $d = 0$.

To get estimation algorithms with fewer queries, we shall use the fact that not every sequence d_1, \dots, d_n is a degree sequence of graphs. For example, if $d_1 = n - 1$,

then necessarily $d_i \geq 1$ for all i . Still, the bad example given above can be modified to show that $\Omega(n/d_0)$ queries are required in order to distinguish between the cases $d \leq d_0$ and $d \geq 2d_0 - O((d_0)^2/n)$. In the first of these two cases we can have all $d_i = d_0$. In the second of these two cases we can have $d_i = d_0$ for all vertices except for d_0 vertices of degree $n - 1$. Hence if we wish to have estimation algorithms with a sublinear (in n) number of queries, we need to restrict ourselves to $\rho \geq 2$.

There is one more restriction that we introduce. Observe that if G contains only one edge, one needs $\Omega(n)$ queries to distinguish this case from $d = 0$. To avoid the problem of handling such very sparse graphs (which are often not interesting anyway), we shall assume that $d \geq d_0$ for some d_0 that will be a parameter of our estimation algorithms. The reader may think of d_0 as typically having value at least 1. Hence the estimation algorithm is allowed to output $d^* = 0$ as an estimation of d for very sparse graphs, even though the ratio between d and d^* is in this case infinite. (The assumption that $d \geq d_0$ can be replaced by allowing the estimation algorithm to have an additive error of d_0 in addition to the multiplicative error of ρ .)

As noted above, for $\rho < 2$ and $d_0 = 1$, the number of queries needed by an estimation algorithm might be $\Omega(n)$. Our main observation is that for $\rho > 2$ and for $d_0 = 1$, the number of queries in the estimation algorithm drops dramatically, from $\Omega(n)$ to $O(\sqrt{n})$. This result is stated in more technical terms in the following theorem.

THEOREM 4. *For any $d_0 > 0$, $\epsilon > 0$, and $\rho = 2 + \epsilon$, there is a ρ estimation algorithm for the average degree of a graph that uses $O(\frac{1}{\epsilon}\sqrt{n/d_0})$ queries, and it is applicable to all graphs of average degree at least d_0 .*

In terms of the application of estimating the average degree in the graph, the more interesting part of our upper bound on the number of queries is the term $\sqrt{n/d_0}$. The dependency on ϵ may be less interesting, especially if one is satisfied with large values of ϵ , such as $\epsilon = 1$. However, achieving a linear dependency in $1/\epsilon$ (rather than some polynomial dependency) is the part that uses Theorem 1.

In section 4 we prove Theorem 4. In section 5 we show how Theorem 4 can be used in order to obtain Theorem 5, addressing a problem that is studied in [3].

THEOREM 5. *There is a randomized algorithm that runs in time $O(\frac{mn \log n}{\epsilon\sqrt{c}})$ on graphs with n vertices and m edges and outputs a list of edges that with high probability satisfies the following:*

1. *Every edge that is on at least c shortest paths is on the list.*
2. *No edge that is on less than $(1/2 - \epsilon)c$ shortest paths is on the list.*

3. Proof of Theorem 1. In this section we prove Theorem 1. Let us first try to clarify our proof plan. It is based on a sequence of transformations whose goal is to simplify the random variables until a case analysis becomes manageable. Known arguments (the *reduce support* operation that will be explained in our proof) show that we may assume that every random variable by itself is “simple” in the sense that it has small support. Hence it is reasonable to expect that if the number of random variables is a small constant, then the theorem (if true) can be proven by a “brute force” case analysis. For example, Samuels [6] mentions 25 cases that are to be considered if one is to prove Conjecture 3 for the case $n = 4$. In our proof we describe a merge operation that replaces two random variables with one random variable. However, we do not perform this merge operation until the number of random variables becomes small, because this operation might create random variables whose mean is larger than 1. After rescaling the random variables, this corresponds to decreasing δ in the statement of Theorem 1 to be arbitrarily small and drives the bound $\delta/(1 + \delta)$ to 0. Instead, we perform the merge operation until a step in which many random variables

may still remain, but they have nice properties. Namely, all random variables, except for perhaps one, have roughly the same mean and, moreover, have small “surplus” (to be defined in the proof). Thereafter, case analysis becomes possible through the use of Proposition 10, which allows us to analyze many such random variables as if they were just one random variable. It is interesting to note that even though our proof plan does not seem to allow us to prove tight bounds (due to the fact that we generate random variables with mean above 1), it does in fact provide optimal bounds when δ is small ($\delta < 1/12$). We shall comment more on our proof in section 3.2, and now we present the proof itself.

Proof. Fix n , δ , and arbitrary nonnegative random variables X_1, \dots, X_n with means at most 1. We prove that inequality (1) holds. We assume without loss of generality that the support of every random variable is composed of a finite set of values. (This is a standard argument, but we sketch it for completeness. Any value larger than $\mu + \delta$ in the support of a random variable can be lowered to $\mu + \delta$ without increasing the probability that $X < \mu + \delta$. Thereafter, any continuous random variable can be approximated by a discrete random variable with the same mean and whose support includes only multiples of ϵ , where ϵ is chosen to be much smaller than δ/n . For these new random variables, X'_1, \dots, X'_n , the event $X' < \mu + \delta'$, where $\delta' = \delta - \epsilon n$, implies that for the original variables, $X < \mu + \delta$. By making ϵ arbitrarily small, we can make δ' arbitrarily close to δ .)

Our proof of inequality (1) consists of a sequence of transformations on the variables X_i . We may view these transformations as occurring in discrete time steps, and in our notation, superscripts will denote time steps. Hence, after time step t , random variables are denoted by X_i^t , their sum by X^t , and the expectation of X^t by μ^t . For $t = 0$, we have the original random variables. All our transformations will have the property that for every $t \geq 0$,

$$(2) \quad \Pr[X^{t+1} < \mu^{t+1} + \delta] \leq \Pr[X^t < \mu^t + \delta].$$

Some properties of the random variables may change by the transformations. In particular, the reduce support transformation (to be defined shortly) when applied to two random variables that were originally identically distributed might transform them to new random variables that are not identically distributed. Moreover, the merge transformation might generate random variables whose mean is larger than 1, even though all original random variables have mean at most 1. We now describe the transformations.

Remove constant. This transformation is applied whenever there is a random variable X_i^t that is constant, that is, $\Pr[X_i^t = \mu_i^t] = 1$. Such a random variable is removed, and $\mu^{t+1} = \mu^t - \mu_i^t$. Clearly, *remove constant* satisfies inequality (2).

Reduce support. This transformation is applied to every random variable whose support has at least three values, and replaces it with a new random variable with the same mean, and whose support includes at most two values from the original support.

LEMMA 6. *Let X_i^t be a random variable whose support includes at least three values. Then X_i^t can be replaced with a new variable X_i^{t+1} with $\mu_i^{t+1} = \mu_i^t$ and whose support includes only two values from the original support of X_i^t . This can be done in a way that satisfies inequality (2).*

Proof. Let $\{v_1, \dots, v_k\}$ be the support of X_i^t , and for $1 \leq j \leq k$, let q_j denote the conditional probability of the event $[X^t < \mu^t + \delta^t]$, conditioned on the event $[X_i^t = v_j]$. For X_i^{t+1} and for $1 \leq j \leq k$, we wish to select $p_j = \Pr[X_i^{t+1} = v_j]$ under the restrictions that the mean of X_i^{t+1} is the same as the mean of X_i^t and that

inequality (2) is satisfied. This can be expressed by the following linear program over the variables p_j :

Minimize $\sum_{j=1}^k q_j p_j$

subject to

- $\sum_{j=1}^k p_j = 1,$
- $\sum_{j=1}^k p_j v_j = \mu_i^t,$
- $p_j \geq 0$ for every $j.$

The above linear program is feasible (as the probabilities associated with the original X_i^t satisfy the constraints). By the theory of linear programming, there is a *basic* optimal solution in which at most two p_j are nonzero. \square

We remark that Lemma 6 has several alternative proofs and in general does not require X_t to have finite support. A similar lemma (with a functional analytic proof) is used in [6].

Align with 0. This transformation is applied to every random variable whose support has two values and these values are greater than 0 (say X_i^t has value v_1 with probability p and v_2 with probability $(1 - p)$, with $0 < v_1 < v_2$) and replaces it with a random variable X_i^{t+1} that has value $v_1 - v_1 = 0$ with probability p and has value $v_2 - v_1$ with probability $(1 - p)$. Hence $\mu_i^{t+1} = \mu_i^t - v_1$, and $\mu^{t+1} = \mu^t - v_1$. Clearly, *align with 0* satisfies inequality (2).

Merge. This transformation takes the two random variables with smallest mean (say X_i^t and X_j^t) and replaces them with a new random variable in three steps. First, replace X_i^t and X_j^t with a new random variable that is distributed like their sum $X_i^t + X_j^t$. Then apply *reduce support* to this new random variable. Finally, apply *align with 0* or *remove constant* to the new random variable (if applicable).

It is easy to see that the transformation *merge* satisfies inequality (2).

The sequence of transformations that we perform is partitioned into two stages. We now describe the first stage.

Stage 1:

1. Whenever possible, apply *remove constant*.
2. Apply *reduce support* until all random variables have support of size at most two. (Different variables may have different support.)
3. Apply *align with 0* to all variables.
4. Apply *merge* until either the number of random variables is reduced to one, or all random variables have mean at least $1/2$ (whichever happens first).

Stage 1 must end because with each application of *merge*, the number of random variables decreases. Let t denote the step after which Stage 1 ends, and let $X_1^t, \dots, X_{n'}^t$ be the random variables that remain. We assume that they are sorted in order of decreasing μ_i^t . Their number n' may be smaller than n , because some of the transformations remove random variables. These are not arbitrary random variables, as each of them has a support of two values, one of which is 0, and the stopping condition for the merge transformations has been reached. For a random variable X_i^t as above, let μ_i^t denote its mean, let $\{0, v_i^t\}$ denote its support, and let $s_i^t = v_i^t - \mu_i^t$ denote its *surplus*. Let $s^t = \sum_{i=1}^{n'} s_i^t$ denote the total surplus.

PROPOSITION 7. *If the total surplus satisfies $s^t < \delta$, then $\Pr[X^t \geq \mu^t + \delta] = 0$.*

Proof. X^t is maximized when $X_i^t = v_i^t$ for all i . In this case,

$$X^t = \sum_{i=1}^{n'} (\mu_i^t + s_i^t) = \mu^t + s^t < \mu^t + \delta. \quad \square$$

Hence we may assume without loss of generality that $s^t \geq \delta$.

LEMMA 8. *If Stage 1 ended with a random variable with mean below 1/2, then $\Pr[X^t < \mu^t] \geq \delta/(1/2 + \delta)$.*

Proof. In this case, exactly one random variable remains. Let X_1^t be the random variable left, with mean $\mu_1^t < 1/2$ and support $\{0, v_1 = \mu_1^t + s^t\}$. Note that the event $X_1^t = 0$ implies $X^t < \mu^t$. Now $\Pr[X_1^t = 0] = s^t/(\mu_1^t + s^t) \geq \delta/(1/2 + \delta)$, because $s^t \geq \delta$ and $\mu_1^t < 1/2$. \square

Observe that Lemma 8 offers a conclusion that is even stronger than that required by Theorem 1, as is illustrated by the following sequence of inequalities:

$$\Pr[X < \mu + \delta] \geq \Pr[X^t < \mu^t + \delta] \geq \Pr[X^t < \mu^t] \geq \frac{\delta}{1/2 + \delta} > \frac{\delta}{1 + \delta}.$$

Hence we may also assume that Stage 1 ended with all random variables having mean at least 1/2. The following property will be used in this case.

PROPOSITION 9. *If Stage 1 ended with all random variables having mean at least 1/2, then $\mu_1^t/2 \leq \mu_{n'}^t \leq \mu_1^t < 3/2$.*

Proof. Recall that the random variables are assumed to be sorted with μ_1^t being the largest mean and $\mu_{n'}^t$ being the smallest mean.

If no random variable has mean greater than 1, then we are done. Hence consider the first time that a random variable with mean greater than 1 is created. This happens by merging two random variables, say at time step r (shortly we will see that in fact it must hold that $r = t - 1$), with the random variables being X_i^r and X_j^r . Let $\mu_i^r \geq \mu_j^r$ be their means before the merge. By the definition of *merge*, no other variable had mean smaller than μ_i^r . By the stopping rule for Stage 1, $\mu_j^r < 1/2$. To get a variable with mean greater than 1, we must have $\mu_i^r > 1/2$. Note that Stage 1 ends after the merge, because no variable with mean below 1/2 is left. Hence the new variable created becomes X_1^{r+1} with $1 < \mu_1^{r+1} < 1 + 1/2 = 3/2$. But as $\mu_1^{r+1} \leq 2\mu_i^r$ and $\mu_{n'}^{r+1} \geq \mu_i^r$, it follows that $\mu_{n'}^{r+1} \geq \mu_1^{r+1}/2$. \square

Let us pause at this point and explain what remains to be proved. All random variables can be assumed to be 2-valued, with one of the values being 0 and with all means μ_i^t satisfying $\mu_1^t/2 \leq \mu_i^t \leq \mu_1^t$. Moreover, the total surplus s^t satisfies $s^t \geq \delta$. For random variables as above we in fact will bound $\Pr[X^t < \mu^t]$ rather than $\Pr[X^t < \mu^t + \delta]$. Lemma 11 (its first part) and Lemma 12 will show that $\Pr[X^t < \mu^t] \geq \min[\delta/(\mu_1^t + \delta), 1/13]$. This almost proves Theorem 1, except that it might happen that at the end of Stage 1, $\mu_1^t > 1$. This possibility is handled in the second part of Lemma 11 by showing that one merge operation before the end of Stage 1 we had $\Pr[X^{t-1} < \mu^{t-1} + \delta] \geq \delta/(1 + \delta)$.

The following proposition is used several times in the proofs of Lemmas 11 and 12. It is most effective when $s < \mu_n$, and μ_n is not much smaller than μ_1 .

PROPOSITION 10. *Let X_1, \dots, X_n be independent random variables with means $\mu_1 \geq \dots \geq \mu_n$ and supports $\{0, \mu_1 + s_1\}, \dots, \{0, \mu_n + s_n\}$, and let $X = \sum_{i=1}^n X_i$, $\mu = \sum_{i=1}^n \mu_i$, and $s = \sum_{i=1}^n s_i$. Then*

$$\Pr[X < \mu - \mu_n + s] \geq \frac{s}{\mu_1 + s}.$$

Proof. It suffices that one random variable comes up 0 to imply $X < \mu + s - \mu_n$. (The inequality is strict because only a variable with $s_i > 0$ may come up 0.) Hence

$$\Pr[X \geq \mu + s - \mu_n] = \prod_{i=1}^n \frac{\mu_i}{\mu_i + s_i} \leq \prod_{i=1}^n \frac{\mu_1}{\mu_1 + s_i}.$$

Given that $\sum_{i=1}^n s_i = s$ and that $s_i \geq 0$, the above product is maximized when $s_1 = s$ and $s_i = 0$ for all $i > 1$, giving $\mu_1/(\mu_1+s)$. Hence $\Pr[X < \mu - \mu_n + s] \geq s/(\mu_1+s)$. \square

The following lemma illustrates the desired outcome of Stage 1.

LEMMA 11.

1. If Stage 1 ended with all random variables having mean at least $1/2$, and if $s^t < \mu_n^t$, then

$$\Pr[X^t < \mu^t] \geq \frac{\delta}{\mu_1^t + \delta} \geq \frac{\delta}{3/2 + \delta}.$$

2. If in addition $\delta \leq 1/12$, then either $\Pr[X^t < \mu^t] < \delta/(1 + \delta)$ or one merge operation before the end of Stage 1 it must have been the case that

$$\Pr[X^{t-1} < \mu^{t-1} + \delta] \geq \delta/(1 + \delta).$$

Remark. The choice of $\delta \leq 1/12$ in the second part of Lemma 11 is made because $\delta/(1 + \delta) = 1/13$ for $\delta = 1/12$. The limiting factor for improving beyond $1/13$ is Lemma 12 rather than Lemma 11. For $\delta > 1/12$ the second part of Lemma 11 simply implies that $\Pr[X^{t-1} < \mu^{t-1} + \delta] \geq \Pr[X^{t-1} < \mu^{t-1} + 1/12] \geq 1/13$.

Proof. The surplus s^t is smaller than the mean of any of the random variables. Using Proposition 10 we then have $\Pr[X^t < \mu^t] \geq \frac{s^t}{\mu_1^t + s^t}$. Using the assumption that $s^t \geq \delta$ and the fact that $\mu_1^t \leq 3/2$ (Proposition 9), we have that $\Pr[X^t < \mu^t] \geq \delta/(3/2 + \delta)$.

To prove the second part of the lemma, note that if it happens that $\mu_1^t \leq 1$, then we have $\Pr[X^t < \mu^t] \geq \delta/(1 + \delta)$. Hence we may assume that $\mu_1^t > 1$, implying in particular that X_1^t is the result of the last merge operation (see the proof of Proposition 9). Let $s' = s^t - s_1^t$ be the surplus of all variables except for X_1^t . If $s' > 0$ (which implies $\mu_2^t \geq 1/2 > 0$), then analysis as in the proof of the first part of the lemma implies that

$$\Pr[X^t \geq \mu^t] \leq \frac{\mu_2^t}{\mu_2^t + s'} \leq \frac{1}{1 + s'}.$$

Hence if $s' \geq \delta$, then $\Pr[X^t < \mu^t] \geq \delta/(1 + \delta)$. So we can assume that $s' < \delta$ (also including the possibility that $s' = 0$).

Let us backtrack to the last merge operation. Hence instead of X_1^t we have two variables X_i^{t-1} and X_j^{t-1} that were merged to give X_1^t . Let their means be $\mu_i^{t-1} \geq \mu_j^{t-1}$ and their surpluses be s_i^{t-1} and s_j^{t-1} . Observe that necessarily $\mu_j^{t-1} < 1/2$ (otherwise the merge operation would not have been performed), and then the assumption that $\mu_1^t > 1$ implies that $\mu_i^{t-1} > 1/2$. As the total surplus of all random variables except for X_i^{t-1} and X_j^{t-1} is $s' < \delta$, we must have $X_i^{t-1} + X_j^{t-1}$ come up larger than $\mu_i^{t-1} + \mu_j^{t-1}$ for $X^{t-1} \geq \mu^{t-1} + \delta$. We consider now two cases.

Case 1. $s_i^{t-1} > 2\delta$. Then $\Pr[X_i^{t-1} = 0] = \frac{s_i^{t-1}}{\mu_i^{t-1} + s_i^{t-1}} \geq \frac{2\delta}{\mu_i^{t-1} + 2\delta}$. If $X_i^{t-1} = 0$, then in order to have $X_i^{t-1} + X_j^{t-1} > \mu_i^{t-1} + \mu_j^{t-1}$ we must have $X_j^{t-1} > \mu_i^{t-1} + \mu_j^{t-1}$. But this happens with probability at most $\frac{\mu_j^{t-1}}{\mu_i^{t-1} + \mu_j^{t-1}} \leq \frac{1/2}{\mu_i^{t-1} + 1/2}$. Hence

$$\Pr[X^{t-1} < \mu^{t-1} + \delta] \geq \frac{2\delta}{\mu_i^{t-1} + 2\delta} \cdot \frac{\mu_i^{t-1}}{\mu_i^{t-1} + 1/2} \geq \frac{\delta}{1 + \delta},$$

where the last inequality holds for $\delta \leq 1/2$ because $1/2 \leq \mu_i^{t-1} \leq 1$.

Case 2. $s_i^{t-1} \leq 2\delta$. Define $s'' = s' + s_i^{t-1}$ as the surplus of all random variables except for s_j^{t-1} , and observe that $s'' < 3\delta \leq 1/4$, the last inequality holding for $\delta \leq 1/12$. We now consider several subcases.

- $s'' < \delta$ and $s_j^{t-1} \geq 1/2$. The fact that $s'' < \delta$ implies that it suffices for X_j^{t-1} to come up 0 to ensure $X^{t-1} < \mu^{t-1} + \delta$. This happens with probability at least $s_j^{t-1}/(\mu_j^{t-1} + s_j^{t-1}) \geq 1/2$.
- $s'' < \delta$ and $s_j^{t-1} < 1/2$. The fact that all random variables except for X_j^{t-1} have mean at least $1/2$ implies that it suffices for one random variable to come up 0 to ensure $X^{t-1} < \mu^{t-1} + \delta$. As necessarily $s'' + s_j^{t-1} \geq \delta$ and $\mu_k^{t-1} \leq 1$ for all k , this happens with probability at least $\delta/(1 + \delta)$ by Proposition 10.
- $\delta \leq s'' < \delta + \mu_j^{t-1}$ and $s_j^{t-1} \geq \delta$. It suffices for X_j^{t-1} to come up 0 to ensure $X^{t-1} < \mu^{t-1} + \delta$. This happens with probability at least $\delta/(1/2 + \delta)$.
- $\delta \leq s'' < \delta + \mu_j^{t-1}$ and $s_j^{t-1} < \delta$. It suffices for some random variable other than X_j^{t-1} to come up 0 to ensure $X^{t-1} < \mu_{t-1} - 1/2 + s'' + \delta < \mu^{t-1} + \delta$. (We used the fact that every random variable except for X_j^{t-1} has mean at least $1/2$ and that $s'' \leq 1/4$.) This happens with probability at least $\delta/(1 + \delta)$ by Proposition 10.
- $s'' \geq \delta + \mu_j^{t-1}$. By Proposition 10, there is probability of at least $\frac{\delta + \mu_j^{t-1}}{1 + \delta + \mu_j^{t-1}}$ for a random variable other than X_j^{t-1} to come up 0. Thereafter X_j^{t-1} must come up at least $\mu_j^{t-1} + \delta + 1/2 - s'' \geq \mu_j^{t-1} + \delta + 1/4$ for $X^{t-1} \geq \mu^{t-1} + \delta$ to hold. The probability of this is at most $\mu_j^{t-1}/(\mu_j^{t-1} + 1/4 + \delta) \leq \mu_j^{t-1}/(\mu_j^{t-1} + 1/4)$. Hence

$$\Pr[X^{t-1} < \mu^{t-1} + \delta] \geq \frac{\delta + \mu_j^{t-1}}{1 + \delta + \mu_j^{t-1}} \cdot \frac{1/4}{\mu_j^{t-1} + 1/4} \geq \frac{\delta}{1 + \delta},$$

where the last inequality holds when $4(\delta + \delta^2 + \mu_j^{t-1}\delta) \leq 1$, which is true for our parameters of $\mu_j^{t-1} \leq 1/2$ and $\delta \leq 1/12$. \square

Summarizing, Lemmas 8 and 11 prove Theorem 1, except for the case that Stage 1 ended with $\mu_1^t/2 \leq \mu_{n'}^t \leq \mu_1^t$ and $s^t \geq \mu_{n'}^t$. We now address this last case. For this case we shall not use the extra slack offered by δ , but rather we shall show the stronger inequality $\Pr[X^t < \mu^t] \geq 1/13$. To prove this last inequality, we perform Stage 2 of our sequence of transformations. It is composed of a modified form of the merge operations, which we call *modified merge*. The modification will allow us to deal with the event $X^t < \mu^t$ rather than $X^t < \mu^t + \delta$. Recall that the *reduce support* operation was based on a linear program that minimized $\Pr[X^{t+1} < \mu^{t+1} + \delta]$ (via the definition of the q_j in the proof of Lemma 6). Modify the *reduce support* operation by modifying the objective function of the linear program to be $\Pr[X^{t+1} < \mu^{t+1}]$ (by making the respective change in the definition of q_j). Use this *modified reduce support* rather than the original *reduce support* as the second step of *modified merge*. Now *modified merge* maintains the inequality

$$(3) \quad \Pr[X^{t+1} < \mu^{t+1}] \leq \Pr[X^t < \mu^t].$$

Note that an application of *modified merge* cannot increase the total surplus. (This was also true for *merge*.) This works in our favor, because as the proof of Lemma 11 demonstrates, it is easier to perform case analysis when the total surplus is small. However, an application of modified merge may result in a random variable

whose mean is smaller than $\mu_1^{t+1}/2$. (For simplicity of notation, we assume that after every step the variables are renamed so as to keep μ_1^{t+1} the largest mean.) This will complicate our case analysis. But note that even with repeated applications of *modified merge*, there will be at most one such random variable. Let us define $s' = \sum s_i$, where the sum is taken over all random variables whose mean is at least $\mu_1/2$. In particular, at the time when Stage 1 ends we may assume that $s' = s^t$ by Proposition 9.

Stage 2. Apply *modified merge* (on the two random variables with currently lowest mean) until a step (that we shall denote by r) after which either the number of remaining (nonconstant) random variables is one or the condition $s' \leq \alpha\mu_1^r$ has been reached for some constant $0 < \alpha < 1/2$ that will be determined later. Stage 2 must eventually end, because with each application of *modified merge* the number of random variables decreases.

LEMMA 12. *Let $\alpha = 1/3$, and let r denote the time step at which Stage 2 ends. Then either $\Pr[X^r < \mu^r] \geq 1/13$ or one modified merge operation before Stage 2 ends $\Pr[X^{r-1} < \mu^{r-1}] \geq 1/13$.*

Proof. The proof of Lemma 12 is based on a case analysis. Some of the details in the case analysis are included so as to get the explicit bound of $1/13$. Those readers who are just interested in verifying that the lemma holds for some universal constant (though perhaps much smaller than $1/13$) may simply think of α as some small constant (say $\alpha = 1/10$) and β (to be introduced shortly) as a much smaller constant (say $\beta = 1/100$) and read each case only up to the point where it becomes clear that under this setting of the parameters; the case in question gives a probability bounded away from 0.

LEMMA 13. *If Stage 2 ends without the condition $s' \leq \alpha\mu_1^r$ being reached, then $\Pr[X^r < \mu^r] > \alpha/(1 + \alpha)$.*

Proof. In this case we have only one nonconstant random variable, X_1^r , with support $\{0, \mu_1^r + s'\}$:

$$\Pr[X^r < \mu^r] = \Pr[X_1^r = 0] = \frac{s'}{\mu_1^r + s'} > \frac{\alpha}{1 + \alpha}. \quad \square$$

LEMMA 14. *If Stage 2 ends with $\beta\mu_1^r \leq s' \leq \alpha\mu_1^r$, where $0 \leq \beta \leq \alpha$ is some constant that will be optimized later, then*

$$\Pr[X^r < \mu^r] \geq \min \left[\left(\frac{\alpha - 2\alpha^2}{1 + \alpha} \right), \left(\frac{\beta - 2\beta^2}{1 + \beta} \right) \right].$$

Proof. Consider first only the random variables with mean at least $\mu_1^r/2$, let X' be their sum, and let μ' be the expectation of X' . Over these random variables, the surplus is $s' = \gamma\mu_1^r$, with $\beta \leq \gamma \leq \alpha < 1/2$. By Proposition 10,

$$\Pr \left[X' < \mu' - \left(\frac{1}{2} - \gamma \right) \mu_1^r \right] \geq \frac{s'}{\mu_1^r + s'} = \frac{\gamma}{1 + \gamma}.$$

The event $X' < \mu' - (1/2 - \gamma)\mu_1^r$ does not yet imply that $X^r < \mu^r$. There still might be one variable $X_{n''}^r$ with $\mu_{n''}^r < \mu_1^r/2$. If $X_{n''}^r$ turns out $\mu_{n''}^r + s_{n''}^r$ and $s_{n''}^r \geq (1/2 - \gamma)\mu_1^r$, then it still may hold that $X^r \geq \mu^r$.

Let us first assume that $\mu_{n''}^r \leq s' = \gamma\mu_1^r$. Then by Markov's inequality,

$$\Pr[X_{n''}^r = \mu_{n''}^r + s_{n''}^r] = \frac{\mu_{n''}^r}{\mu_{n''}^r + s_{n''}^r} \leq \frac{\gamma\mu_1^r}{\gamma\mu_1^r + (1/2 - \gamma)\mu_1^r} = 2\gamma.$$

Hence

$$\Pr[X^r < \mu^r] \geq \frac{\gamma}{1 + \gamma} \cdot (1 - 2\gamma) = \frac{\gamma - 2\gamma^2}{1 + \gamma}.$$

For $0 < \beta \leq \gamma \leq \alpha < 1/2$, the expression above is minimized when $\gamma \in \{\alpha, \beta\}$.

We are left with the case that $\mu_{n''} > s'$. But then we have

$$\Pr[X^r < \mu^r] \geq \Pr[X_{n''}^r = 0] = \frac{s_{n''}^r}{\mu_{n''}^r + s_{n''}^r} \geq \frac{1/2 - \gamma}{1 - \gamma},$$

where we have used the fact that $\mu_{n''}^r < \mu_1^r/2$ and $s_{n''}^r \geq (1/2 - \gamma)\mu_1^r$. As $\gamma \leq \alpha$, we have that $\Pr[X^r < \mu^r] \geq (1/2 - \alpha)/(1 - \alpha)$. But this probability is larger than $(\alpha - 2\alpha^2)/(1 + \alpha)$ of the previous case, and hence can be ignored. \square

LEMMA 15. *If Stage 2 ends with $s' < \beta\mu_1^r$, and $0 < \beta < \alpha/2$, then one merge prior to the end of Stage 2 it must have been the case that $\Pr[X^{r-1} < \mu^{r-1}]$ was at least the minimum of the following expressions:*

1. $\frac{\alpha - \beta}{1/2 + \alpha - \beta} \cdot \frac{1/2 - \beta}{1 - \beta}$.
2. $\frac{\alpha - 3\beta/2}{1 + \alpha - 3\beta/2} \cdot \frac{1 - 3\beta/2}{3/2 - 3\beta/2}$.
3. $\frac{\alpha - 2\beta}{1 + \alpha - 2\beta}$.
4. $\left(\frac{1/2 - \beta}{1 - \beta}\right)^2$.
5. $\frac{1/2 - 3\beta/2}{3/2 - 3\beta/2} \cdot \frac{1 - 3\beta/2}{3/2 - 3\beta/2}$.

The proof of Lemma 15 involves a detailed case analysis and appears in section 3.1.

Summing up, Lemmas 14 and 15 imply that after Stage 2, either $\Pr[X^r < \mu^r]$ or $\Pr[X^{r-1} < \mu^{r-1}]$ is at least the smallest of the following quantities (where $0 < \beta < \alpha/2 < 1/4$):

- $\frac{\alpha - 2\alpha^2}{1 + \alpha}$;
- $\frac{\beta - 2\beta^2}{1 + \beta}$;
- $\frac{\alpha - \beta}{1/2 + \alpha - \beta} \cdot \frac{1/2 - \beta}{1 - \beta}$;
- $\frac{\alpha - 3\beta/2}{1 + \alpha - 3\beta/2} \cdot \frac{1 - 3\beta/2}{3/2 - 3\beta/2}$;
- $\frac{\alpha - 2\beta}{1 + \alpha - 2\beta}$;
- $\left(\frac{1/2 - \beta}{1 - \beta}\right)^2$;
- $\frac{1/2 - 3\beta/2}{3/2 - 3\beta/2} \cdot \frac{1 - 3\beta/2}{3/2 - 3\beta/2}$.

Choosing (suboptimally) $\alpha = 1/3$ and $\beta = 1/8$ gives at least $1/13$ in all cases.

This completes the proof of Lemma 12. \square

We can now summarize the proof of Theorem 1. We have the original random variables for which we wish to prove $\Pr[X < \mu + \delta] \geq \min[\delta/(1 + \delta), 1/13]$. The proof proceeds in two stages. In Stage 1 we apply a sequence of transformations maintaining the inequality (2) until a step t in which all random variables have mean at least $1/2$ (or only one random variable remains, which is an easy case to handle). Then Lemma 11 implies the theorem whenever the total surplus s^t is small ($s^t < \mu_{n'}^t$). To handle the case that the surplus is large, we perform a sequence of transformations in Stage 2, this time maintaining the inequality (3), until a step r in which the surplus of all variables (except for the variable with smallest mean) is no longer large compared to the maximum mean (namely, $s' \leq \alpha\mu_1^r$). Then Lemma 12 implies that also in this case (corresponding to the case that s^t was greater than $\mu_{n'}^t$) the theorem holds. Note

that, overall, depending on which case is considered, Lemmas 11 and 12 do not prove inequality (1) on the original random variables but on transformed random variables that appear after one of the steps $t - 1, t, r - 1$, or r . But this implies inequality (1) on the original random variables, because of the inequalities (2) and (3). \square

3.1. Remaining analysis for Stage 2. We prove here Lemma 15, whose proof was the only part omitted from the proof of Theorem 1.

Proof. Consider the last two random variables to have been merged, say X_i^{r-1} and X_j^{r-1} , with means $\mu_i^{r-1} \geq \mu_j^{r-1}$, and let μ_1^{r-1} be the largest mean at time $r - 1$. After the *modified merge* of X_i^{r-1} and X_j^{r-1} , the largest mean μ_1^r may still have been μ_1^{r-1} , but it could also be as high as $\mu_i^{r-1} + \mu_j^{r-1}$ if this happens to be higher than μ_1^{r-1} . In fact, μ_1^r may also be lower than μ_1^{r-1} if only one variable is left at the end of Stage 2 and this variable underwent an *align with 0* operation. However, in this case the bounds that we get for $X^{r-1} < \mu^{r-1}$ are much stronger than what we get otherwise (details omitted), so we shall ignore this case.

We analyze the situation one merge operation before the end of Stage 2. Note that we know that at that time, $s' \geq \alpha\mu_1^{r-1}$, because otherwise Stage 2 would have ended earlier. Likewise, the sum $\sum s_i^{r-1}$ taken over all variables except X_i^{r-1} and X_j^{r-1} is at most $\max[\beta\mu_1^{r-1}, \beta(\mu_i^{r-1} + \mu_j^{r-1})]$, because otherwise we could not have had $s' < \beta\mu_1^r$ at the end of Stage 2. We consider now two cases.

Case 1. $\mu_j^{r-1} < \mu_1^{r-1}/2$. Hence X_j^{r-1} did not contribute to s' at time step $r - 1$. Note that $\mu_i^{r-1} \geq \mu_1^{r-1}/2$, and hence X_i^{r-1} did contribute to s' at time step $r - 1$. It follows that $s_i^{r-1} \geq \alpha\mu_1^{r-1} - \beta\mu_1^r$. (This last expression is positive, because Lemma 15 assumes that $\beta < \alpha/2$, and $\mu^r \leq 2\mu^{r-1}$.) Hence,

$$\Pr[X_i^{r-1} = 0] \geq \frac{\alpha\mu_1^{r-1} - \beta\mu_1^r}{\mu_i^{r-1} + \alpha\mu_1^{r-1} - \beta\mu_1^r}.$$

If $X_i^{r-1} = 0$, then in order to have $X^{r-1} \geq \mu^{r-1}$, X_j^{r-1} must contribute at least $\mu_j^{r-1} + \mu_i^{r-1} - \beta\mu_1^r$ to X^{r-1} . (The expression $\mu_i^{r-1} - \beta\mu_1^r$ is positive because $\beta < 1/4$ and $\mu_i^{r-1} \geq \mu_1^{r-1}/2 \geq \mu_1^r/4$.) This may happen with probability at most $\mu_j^{r-1}/(\mu_j^{r-1} + \mu_i^{r-1} - \beta\mu_1^r)$. We then have

$$\Pr[X^{r-1} < \mu^{r-1}] \geq \frac{\alpha\mu_1^{r-1} - \beta\mu_1^r}{\mu_i^{r-1} + \alpha\mu_1^{r-1} - \beta\mu_1^r} \cdot \frac{\mu_i^{r-1} - \beta\mu_1^r}{\mu_j^{r-1} + \mu_i^{r-1} - \beta\mu_1^r}.$$

The above expression is minimized when μ_j^{r-1} is maximized, namely, when $\mu_j^{r-1} = \mu_1^{r-1}/2$. As $\mu_i^{r-1} \geq \mu_1^{r-1}/2$, it follows that $\mu_i^{r-1} + \mu_j^{r-1} \geq \mu_1^{r-1}$. The expression above is minimized when μ_1^r is maximized, namely, $\mu_1^r = \mu_j^{r-1} + \mu_i^{r-1}$. Normalizing μ_1^{r-1} to 1 and using the notation μ_i to denote μ_i^{r-1}/μ_1^{r-1} , we have after some rearrangements

$$\Pr[X^{r-1} < \mu^{r-1}] \geq \frac{\alpha - \beta/2 - \beta\mu_i}{(1 - \beta)\mu_i + \alpha - \beta/2} \cdot \frac{(1 - \beta)\mu_i - \beta/2}{(1 - \beta)\mu_i + 1/2 - \beta/2}.$$

The expression above is defined for all $\mu_i \geq 0$. It equals 0 for $\mu_i \in \{\beta/2(1 - \beta), (\alpha - \beta/2)/\beta\}$ and is positive in between. Moreover, there are at most two points where the derivative with respect to μ_i of this expression vanishes (as it is a ratio of two nonproportional quadratics), and for $\beta < 2\alpha/3$ the expression is positive in the allowed

range of $1/2 \leq \mu_i \leq 1$. It follows that the expression is minimized when $\mu_i \in \{1/2, 1\}$, giving

$$\Pr[X^{r-1} < \mu^{r-1}] \geq \min \left[\frac{\alpha - \beta}{1/2 + \alpha - \beta} \cdot \frac{1/2 - \beta}{1 - \beta}, \frac{\alpha - 3\beta/2}{1 + \alpha - 3\beta/2} \cdot \frac{1 - 3\beta/2}{3/2 - 3\beta/2} \right].$$

This gives items 1 and 2 in the statement of Lemma 15.

Case 2. $\mu_j^{r-1} \geq \mu_1^{r-1}/2$. Hence both s_i^{r-1} and s_j^{r-1} did contribute to s' (before the last merge), and moreover, $\mu_i^{r-1} + \mu_j^{r-1} \geq \mu_1^{r-1}$. This, together with the inequality $\mu_1^r \leq \max[\mu_1^{r-1}, \mu_i^{r-1} + \mu_j^{r-1}]$, implies that we may use $\mu_i^{r-1} + \mu_j^{r-1}$ as an upper bound on μ_1^r . To simplify notation and without loss of generality we may assume that $\mu_1^{r-1} = 1$, and then $1/2 \leq \mu_j^{r-1} \leq \mu_i^{r-1} \leq 1$. We have that $s_i^{r-1} + s_j^{r-1} > \alpha - \beta(\mu_i^{r-1} + \mu_j^{r-1}) \geq \alpha - 2\beta$. Recall (from the paragraph prior to Case 1) the sum $\sum s_i^{r-1}$ taken over all variables except X_i^{r-1} and X_j^{r-1} is at most $\beta(\mu_i^{r-1} + \mu_j^{r-1})$. Hence if $X_i^{r-1} + X_j^{r-1} < \mu_i^{r-1} + \mu_j^{r-1} - \beta(\mu_i^{r-1} + \mu_j^{r-1})$, then necessarily $X^{r-1} < \mu^{r-1}$. We let B denote the event $[X_i^{r-1} + X_j^{r-1} < (1 - \beta)(\mu_i^{r-1} + \mu_j^{r-1})]$ and perform a subcase analysis for $\Pr[B]$. The subcases are partitioned according to which of X_i^{r-1} and X_j^{r-1} (or both, or either one) needs to come up 0 in order for B to hold.

1. It suffices that either $X_i^{r-1} = 0$ or $X_j^{r-1} = 0$ for B to hold. In this case, using $\mu_i^{r-1} \leq 1$ and $s_i^{r-1} + s_j^{r-1} \geq \alpha - 2\beta$, Proposition 10 implies that

$$\Pr[B] \geq \frac{\alpha - 2\beta}{1 + \alpha - 2\beta}.$$

This gives item 3 in the statement of Lemma 15.

2. B holds iff $X_i^{r-1} = 0$. In this subcase, necessarily $\mu_i^{r-1} + s_i^{r-1} \geq (1 - \beta)(\mu_i^{r-1} + \mu_j^{r-1})$. Using the fact that $\mu_i^{r-1} \leq 1$ and $\mu_j^{r-1} \geq 1/2$ we have that $s_i^{r-1} \geq \mu_j^{r-1} - \beta\mu_i^{r-1} - \beta\mu_j^{r-1} \geq 1/2 - 3\beta/2$, and therefore

$$\Pr[B] = \Pr[X_i^{r-1} = 0] = \frac{s_i^{r-1}}{\mu_i^{r-1} + s_i^{r-1}} \geq \frac{1/2 - 3\beta/2}{3/2(1 - \beta)}.$$

This subcase is dominated by the subcase above, and hence can be ignored.

3. B holds iff $X_j^{r-1} = 0$. This subcase is analogous to and dominated by the subcase above, and hence can be ignored.
4. B holds only if both $X_i^{r-1} = 0$ and $X_j^{r-1} = 0$. Then necessarily $\mu_i^{r-1} + s_i^{r-1} \geq (1 - \beta)(\mu_i^{r-1} + \mu_j^{r-1})$ and $\mu_j^{r-1} + s_j^{r-1} \geq (1 - \beta)(\mu_i^{r-1} + \mu_j^{r-1})$. We have

$$\Pr[B] \geq \left(\frac{(1 - \beta)(\mu_i^{r-1} + \mu_j^{r-1}) - \mu_i^{r-1}}{(1 - \beta)(\mu_i^{r-1} + \mu_j^{r-1})} \right) \cdot \left(\frac{(1 - \beta)(\mu_i^{r-1} + \mu_j^{r-1}) - \mu_j^{r-1}}{(1 - \beta)(\mu_i^{r-1} + \mu_j^{r-1})} \right).$$

For fixed $\mu_i^{r-1} + \mu_j^{r-1}$ this expression is minimized when $\mu_i^{r-1} - \mu_j^{r-1}$ is maximized. Hence either $\mu_i^{r-1} = 1$ or $\mu_j^{r-1} = 1/2$. Thereafter, it can

be verified that the expression is minimized when the other mean is either maximized or minimized, giving us three possible local minimum points, $\mu_i^{r-1}, \mu_j^{r-1} \in \{1/2, 1\}$, $\mu_j^{r-1} \leq \mu_i^{r-1}$. Two of these give identical values (the cases that $\mu_i^{r-1} = \mu_j^{r-1}$); hence we obtain

$$\Pr[B] \geq \min \left[\left(\frac{1/2 - \beta}{1 - \beta} \right)^2, \left(\frac{1/2 - 3\beta/2}{3/2 - 3\beta/2} \cdot \frac{1 - 3\beta/2}{3/2 - 3\beta/2} \right) \right].$$

This gives items 4 and 5 in the statement of Lemma 15. \square

3.2. Some comments. It is straightforward to modify inequality (1) so that there is no formal requirement that the random variables are nonnegative or that their mean is bounded by 1. Let w be the maximum over all random variables X_1, \dots, X_n of the respective $\mu_i - l_i$, where l_i is the lowest value in the support of X_i . Then

$$(4) \quad \Pr[X \leq \mu + \delta w] \geq \min[\delta/(1 + \delta), 1/13].$$

The constant 1/13 in Theorem 1 is not best possible and can be improved with more detailed case analysis. We suspect that the true constant should be $1/e$. Presumably, the way to prove a tight result is to find a sequence of transformations on the random variables that does not increase $\Pr[X < \mu + \delta]$ and that gradually brings them to the conjectured worst case for $[X < \mu + \delta]$. The sequence of transformations performed in our proof of Theorem 1 manages to achieve this only when $\delta \leq 1/12$ (or some other constant not far from 1/12). However, it fails to characterize the worst case for the perhaps more interesting $\delta = 1$. The idea in the proof is to transform the random variables into a situation where a case analysis becomes manageable, at the possible cost of giving up the tightness of the bound. The main principles used are reducing the support of every random variable to two values, getting all random variables (perhaps except one) to have roughly the same mean, reducing the surplus to be of order of magnitude comparable to this mean, and extracting from arbitrarily many random variables a single event of interest, as done in Proposition 10. It should be clear to the reader that more detailed case analysis would provide tighter results. But let us point out some limitations that relate to Lemma 12. As long as one chooses α not larger than $\mu_1/2$ (and, in fact, not larger than $3\mu_1/2$) and analyzes only the situation at the end of Stage 2 or one step earlier, one cannot obtain a bound better than $\Pr[X < \mu] \geq 2/9$. For example, assume that during Stage 2 we are left with three variables, each with support $\{0, \mu\}$ and mean $\mu/3$. At this point, $\Pr[X < \mu] = (2/3)^3 = 8/27 < 1/e$. After a merge operation, this probability decreases further to $(1/3) \cdot (2/3) = 2/9$. One merge operation later, Stage 2 ends. Hence to get (nearly) tight results using the current approach, one may need to modify the definition of Stage 2 and perform much more extensive (possibly computer assisted) case analysis.

4. Proof of Theorem 4. The reader is assumed to be familiar with elementary methods in probability (such as the use of Markov’s inequality, Chebyshev’s inequality, and Chernoff bounds). If needed, see details in [1], for example.

We query random t vertices and obtain their degrees. Let d_i be the degree returned by the i th query. Basically, our estimator for d will be $d^* = \frac{1}{t} \sum_{i=1}^t d_i$. In section 4.3 we shall modify this estimator so as to improve its quality. For simplicity of analysis, we assume that sampling is done with replacement. (The same vertex might be queried more than once.) This is insignificant when t is small (e.g., $t \leq \sqrt{n}$),

though note that for large values of t (and, in particular, when $t = n$) sampling without replacement gives better estimates than sampling with replacement.

Note that the expectation of our estimate satisfies

$$(5) \quad E[d^*] = d.$$

Hence the estimator is *unbiased*. In deviations from the expectation, we will analyze separately the events $d^* > d$ and $d^* < d$, or rather, $d^* < d/2$.

4.1. The estimate is not too high. Here we shall use Theorem 1. As an immediate consequence of this theorem (taking $\delta = 1$ and using the fact that the degree of a sampled vertex is a nonnegative random variable with expectation d) we have the following corollary.

COROLLARY 16. *There is some universal constant $\alpha > 0$ such that for every graph with average degree d , by querying t random vertices (with replacement) for their degree, the average d^* satisfies $\Pr[d^* \leq (1 + 1/t)d] \geq \alpha$.*

We can take $\alpha = 1/13$ in Corollary 16, and we conjecture that the corollary is also true with $\alpha = 1/e$.

4.2. The estimate is not too low. We assume that the average degree in the graph is at least d_0 . Our sampling algorithm queries $t = k\sqrt{n/d_0}$ vertices at random and reports the sum of the degrees. Here k is a parameter that will later be chosen to be of order $1/\epsilon$.

Let X_i be the random variable that denotes the degree of the i th query, and let $X = \sum_{i=1}^t X_i$. Then $E[X] = t \cdot d$. The following lemma shows that the typical value of X is not much smaller than $E[X]/2$.

LEMMA 17. *For arbitrary $\delta > 0$ (that will later be fixed to $50\sqrt{2}/\alpha$, where α is as in Corollary 16), with probability at least $1 - 4\sqrt{2}/\delta - 2^{-\Omega(\delta)}$,*

$$X \geq \frac{E[X]}{2} \left(1 - \frac{\delta}{k}\right).$$

Proof. Essentially, the proof of the lemma is based on Chebyshev's inequality. To apply Chebyshev's inequality directly, one would need the variance of X to be small compared to $(E[X])^2$. Unfortunately, vertices of very high degree may cause the variance to exceed $(E[X])^2$. To overcome this problem we observe (and will soon show formally) that in every graph, the vertices of very high degree contain at most slightly more than half the endpoints of the edges. The contribution to X of vertices whose degree is not very high is concentrated around its mean, because for them the variance is small. This explains why the value of X is likely to be not much smaller than $E[X]/2$. We now present our proof of the lemma, based on the above principles.

Partition the set of vertices of G into two sets, H (for high) and L (for low). For a constant c (independent of n, d, k) that will be determined later, the set H contains the $c\sqrt{nd}/k$ vertices of highest degree (breaking ties arbitrarily). The set L contains the other vertices. Every edge has two endpoints. Let us partition the endpoints of edges into the following four sets:

- $E_{H,L}$ (the endpoints in H of edges between H and L);
- $E_{L,H}$ (the endpoints in L of edges between H and L);
- $E_{H,H}$ (the endpoints in H of edges between H and H);
- $E_{L,L}$ (the endpoints in L of edges between L and L).

Observe that $|E_{H,H}| \leq |H|^2 = c^2 nd/k^2$. It will be the case that c is a universal constant, whereas $k \geq \Omega(1/\epsilon)$, and hence $|E_{H,H}| = O(\epsilon^2 nd)$. Moreover, we allow an

error of $\epsilon \cdot nd$ in our estimation of nd . Hence, $E_{H,H}$ has only a low order effect on the accuracy of the estimation. So as to simplify notation and the presentation, we shall simply assume that $|E_{H,H}| = 0$. We shall not give a rigorous proof that this assumption has only a low order effect on our analysis but merely note here that formalists may redo the analysis without assuming that $|E_{H,H}| = 0$, and at worst this will affect some constants that are eventually hidden by the O notation.

Let $m_1 = |E_{H,L}|$, $m_2 = |E_{L,H}|$, and $m_3 = |E_{L,L}|$. Hence $m_1 + m_2 + m_3 = dn$. Note that $m_2 = m_1$, because $|E_{L,H}| = |E_{H,L}|$. Let us break the random variable X into the sum of three random variables, $X = Y_1 + Y_2 + Y_3$, according to the contribution to X from m_1 , m_2 , and m_3 , respectively. Let h denote the minimum degree of a vertex in H .

PROPOSITION 18. *With probability $1 - 2^{-\Omega(c)}$,*

$$Y_1 \geq ch/2.$$

Proof. The expected number of vertices queried from H is $t|H|/n = k\sqrt{n/d_0} \cdot c\sqrt{d_0n}/kn = c$. With probability $1 - 2^{-\Omega(c)}$, the actual number of vertices queried from H is at least $c/2$. Each such vertex contributes at least h to Y_1 . \square

PROPOSITION 19. *A vertex in L can cover at most $|H| = c\sqrt{nd_0}/k$ endpoints in $E_{L,H}$.*

Proof. For every endpoint in $E_{L,H}$ covered by a vertex in L , the other endpoint of the respective edge is in H . As the original graph is a simple graph with no parallel edges, the proof follows. \square

PROPOSITION 20. *For $\lambda > 0$, with probability at least $1 - 1/\lambda^2$,*

$$Y_2 \geq E[Y_2] - \lambda\sqrt{cdn/2}.$$

Proof. The variance of Y_2 is maximized if the endpoints of $E_{L,H}$ are concentrated on $m_2/|H|$ vertices (each covering $|H|$ endpoints). Hence

$$\text{var}[Y_2] \leq |H|^2 \frac{m_2}{n|H|} t = cm_2 \leq cdn/2.$$

The proof now follows from Chebyshev's inequality. \square

PROPOSITION 21. *For $\lambda > 0$, with probability at least $1 - 1/\lambda^2$,*

$$Y_3 \geq E[Y_3] - \lambda\sqrt{\frac{hkm_3}{\sqrt{d_0n}}}.$$

Proof. The maximum degree of any vertex in L is h . Hence the graph induced by the edges $E_{L,L}$ also has maximum degree at most h . Thus

$$\text{var}[Y_3] \leq h^2 \frac{m_3}{h} \frac{t}{n} = hm_3k/\sqrt{d_0n}.$$

The proof now follows from Chebyshev's inequality. \square

PROPOSITION 22. *With probability at least $1 - 2/\lambda^2 - 2^{-\Omega(c)}$,*

$$X \geq \frac{E[X]}{2} + \frac{ch}{2} - \lambda\sqrt{\frac{cdn}{2}} - \lambda\sqrt{\frac{hkm_3}{\sqrt{d_0n}}} + \frac{km_3}{2\sqrt{d_0n}}.$$

Proof. $X = Y_1 + Y_2 + Y_3$. By Propositions 18, 20, and 21 we have that with probability at least $1 - 2/\lambda^2 - 2^{-\Omega(c)}$,

$$X \geq E[Y_2] + E[Y_3] + \frac{ch}{2} - \lambda\sqrt{\frac{cdn}{2}} - \lambda\sqrt{\frac{hkm_3}{\sqrt{d_0n}}}.$$

As $E[Y_1] = E[Y_2]$, we have that $E[X]/2 = E[Y_2] + E[Y_3]/2$. Using $E[Y_3] = m_3t/n = km_3/\sqrt{d_0n}$, the proof follows. \square

Fix $c = 4\lambda^2$. Then

$$\frac{ch}{2} \cdot \frac{km_3}{2\sqrt{d_0n}} \geq \left(\lambda\sqrt{\frac{hkm_3}{\sqrt{d_0n}}} \right)^2,$$

implying

$$\frac{ch}{2} - \lambda\sqrt{\frac{hkm_3}{\sqrt{d_0n}}} + \frac{km_3}{2\sqrt{d_0n}} \geq 0.$$

Hence for $c = 4\lambda^2$, the inequality in Proposition 22 can be replaced with

$$X \geq \frac{E[X]}{2} - \lambda\sqrt{\frac{cdn}{2}}.$$

The term $\lambda\sqrt{\frac{cdn}{2}} = \lambda^2\sqrt{2dn}$ is at most $\frac{E[X]}{2}2\lambda^2\sqrt{2}/k$, because $E[X] = dt = k\sqrt{nd}\sqrt{d/d_0}$. Renaming $2\sqrt{2}\lambda^2$ by δ , Lemma 17 is proved. \square

4.3. Combining the upper and lower bound. Let us set $k = 3\delta/\epsilon$ (where ϵ is taken from Theorem 4), and hence from Lemma 17 we have that with probability at least $1 - 4\sqrt{2}/\delta - 2^{-\Omega(\delta)}$,

$$X \geq \frac{E[X]}{2} \left(1 - \frac{\epsilon}{3} \right).$$

By Corollary 16, we have that with probability at least α , $X \leq E[X](1 + 1/t)$. The ratio between the upper bound on X and the lower bound on X is $2(1+1/t)/(1-\epsilon/3) \leq 2 + \epsilon$. This last inequality holds when ϵ is sufficiently small (which implies Theorem 4 also for larger values of ϵ) and t is sufficiently large compared to $1/\epsilon$ (which is true in our context because t is a parameter that grows with the number of vertices n , and the O notation in the statement of Theorem 4 implies that it suffices to prove the theorem when n is sufficiently large).

An *unbiased estimate* consists of taking t samples and returning their sum X . Perform $2/\alpha$ independent unbiased estimates for X , where α is taken to be as in the discussion following Corollary 16. Our estimation procedure returns X_{\min} , the minimum of these estimates. (Equivalently, we set $d^* = X_{\min}/t$.)

$$\Pr \left[X_{\min} \leq E[X] \left(1 + \frac{1}{t} \right) \right] \geq 1 - (1 - \alpha)^{2/\alpha} \geq 1 - \frac{1}{e^2} \geq \frac{5}{6},$$

$$\Pr \left[X_{\min} \geq \frac{E[X]}{2} \left(1 - \frac{\epsilon}{3} \right) \right] \geq 1 - \frac{2}{\alpha} \left(\frac{4\sqrt{2}}{\delta} + 2^{-\Omega(\delta)} \right) \geq \frac{5}{6},$$

where the last inequality uses $\delta = 50\sqrt{2}/\alpha$. This gives $k = 3\delta/\epsilon < 220/\alpha\epsilon$. The total number of queries used in our estimation procedure is $2t/\alpha$. This gives the following corollary.

COROLLARY 23. *For some universal constant β , using*

$$\beta \frac{\sqrt{n/d_0}}{\epsilon}$$

queries, one can estimate the average degree d of an n node graph within a ratio of $(2 + \epsilon)$, provided that $d > d_0$.

Proof. Setting $\beta = (\frac{2}{\alpha})(\frac{220}{\alpha}) = 440/\alpha^2$, we perform $2/\alpha$ unbiased estimates, each with $t = 220\sqrt{n/d_0}\alpha^{-1}\epsilon^{-1}$ queries, and take the minimum of the estimations that they give. \square

Let us note here the role of Corollary 16. It allows us to substitute a universal constant for α (which is shown to be at least $1/13$ in Theorem 1, though we conjecture that $1/e$ also works). If not for Theorem 1, we could have used Markov's inequality in a proof of a modified Corollary 16, showing (for example) that $\Pr[d^* \leq (1+\epsilon/3)d] \geq \epsilon/4$. This would have been equivalent to replacing α in the proof of Corollary 23 with $\epsilon/4$, which would require the number of queries used by the estimation procedure to be $\beta\sqrt{n/d_0}\epsilon^{-3}$ (for some constant β). This a factor of ϵ^{-2} worse than the bounds that we get through the use of Theorem 1.

4.4. Optimality of sample size. The sample size in Corollary 23 is essentially best possible, as the following proposition shows.

PROPOSITION 24. *For every (reasonable) n, d, ϵ , one can construct a graph G_1 with $(1 + \epsilon)nd$ edges and a graph G_2 with $dn/2$ edges, such that $\Omega(\frac{1}{\epsilon}\sqrt{n/d})$ vertices need to be queried in order to have probability above $2/3$ of distinguishing between them.*

Proof. Graph G_1 has a set A of $\epsilon\sqrt{nd}$ vertices of degree $(1 + \epsilon)\sqrt{nd}/\epsilon$ and a set B of $(1 + \epsilon)\sqrt{nd}/\epsilon$ vertices of degree $\epsilon\sqrt{nd}$ (e.g., arranged as a complete bipartite subgraph between A and B). The other vertices have degree 0. Graph G_2 has a set C of \sqrt{nd}/ϵ vertices of degree $\epsilon\sqrt{nd}$.

We sketch the proof of why $\Omega(\sqrt{\frac{n}{d}}\epsilon^{-1})$ queries are necessary. Assume that the number of queries is $\sqrt{\frac{n}{d}}\epsilon^{-1}$. Then there is constant probability that no vertex from A is queried, and the expected number of vertices queried from B is $\epsilon^{-2} + \epsilon^{-1}$. The expected vertices queried from C is ϵ^{-2} . As the standard deviation is of order $\sqrt{\epsilon^{-2}} = \epsilon^{-1}$, there is constant probability that G_1 and G_2 will be confused. \square

The optimality of the sample size was proved under the assumption that the only information used by the estimation algorithm is the degree of the queried vertices. More generally, one may think of randomized estimation algorithms that make use of additional information. For example, when querying a vertex of positive degree, the next vertex to query may be chosen at random from the list of neighbors of the current vertex. The use of a more general class of random estimation algorithms may allow either quicker or more accurate estimation of the average degree in a graph. See [5], for example. However, let us explain here some of the advantages of “degree only” sampling, advantages that might be lost by other estimation algorithms.

1. All queries can be made in parallel, which in some contexts results in saving time.
2. Sampling can be done anonymously. The estimation algorithm need not know the identity of queried vertices nor the identity of their neighbors. Privacy issues may sometimes require that this be the case. For example, vertices of

a graph may represent persons in some community, and an edge may represent some sort of interaction that took place between the respective persons. Persons may be willing to fill out an anonymous questionnaire stating with how many different persons they had interaction (namely, their degree) but may not be willing to disclose with whom they had interaction.

3. In section 5 there are several different graphs G_e defined on the same set of vertices, and in a single degree query one gets the degrees of the respective vertex in all graphs simultaneously. In order to efficiently estimate the average degree in all graphs, it is useful to have an estimation algorithm for which the choice of which vertex to query does not depend on the graph in question.

5. Quickly estimating the load on a network. We have seen how to estimate the average degree in a graph using a relatively small number of degree queries. Graph problems are often abstractions of other more concrete problems. As an example (which motivated this study), consider the following problem motivated and studied in [3].

The input is a connected network G with n vertices and m edges (namely, a graph). Between every two vertices there is a shortest path (a path that crosses the smallest number of edges). We assume here that shortest paths are unique, a point that we shall return to later. For an integer parameter c (that may depend on n), we wish to find all edges that are members of at least c shortest paths. In the terminology of [3], these edges are called “weakest links,” apparently because these are the edges where failure may cause the largest amount of damage to the performance of the network. Finding all weakest links can be done in time $O(nm)$ using an algorithm for all pairs’ shortest paths. The goal in [3] is to do better. They propose a randomized algorithm that with high probability has the following guarantees:

- *Finds weakest links.* It outputs all edges that belong to at least c shortest paths.
- *Avoids false alarms.* It does not output any edge that is a member of less than $(1 - \epsilon)c$ shortest paths.

The running time of the algorithm in [3] is $O(\frac{mn^2 \log n}{c\epsilon^2})$, which is better than that of all pairs’ shortest paths when $c \gg n \log n$. The basic idea in this algorithm is to choose $k \simeq \frac{n^2 \log n}{c\epsilon^2}$ pairs of vertices at random and for each pair to perform a shortest path computation (taking $O(m)$ operations per pair). Using the collection of k shortest paths that are found, one estimates in how many shortest paths each edge participates.

Here we present a faster algorithm for finding the weakest links. It is based on two observations. One is that the cost of performing single source shortest path computations (namely, that of finding the shortest paths from one vertex to all other vertices) is $O(m)$, similar to that of finding the shortest path between one pair of vertices. The other observation is that the estimation problem that this gives rise to can be cast as that of estimating the average degree in a graph or, more precisely, in m different graphs simultaneously. The improved running time comes at a cost of a somewhat weaker guarantee in terms of false alarms:

- *Avoids false alarms.* The algorithm does not output any edge that is a member of less than $(1/2 - \epsilon)c$ shortest paths.

As in [3] we assume that shortest paths are unique. This requires a convention for breaking ties between paths of equal length. We shall use the same convention that is proposed in [3], namely, to take the lexicographically first such path. This convention assumes that vertices are labelled and that there is a total ordering on the labels. For

example, the labels can be 1 to n . A path can be viewed as a sequence of vertices in a natural way. Hence a path is a sequence of labels. In fact, two sequences correspond to the same path, depending on which of its two endpoints is considered to be the head of the path and which is considered to be the tail. The name of the path is taken to be the lexicographically smaller of the two. Given two different paths that connect the same pair of vertices, if they are of equal length we use the convention that the one with the lexicographically smaller name is considered to be shorter.

PROPOSITION 25. *Under the tiebreaking convention specified above, there is an $O(m)$ -time algorithm that does the following. Given a connected graph G with n vertices and m edges and an arbitrary vertex v , it simultaneously counts for every edge e in how many vertices u edge e participates in the shortest path connecting u and v .*

Proof. We assume a model of computation in which algorithms such as single source shortest path take $O(m)$ time. In particular, some basic operations (such as comparison between two $O(\log n)$ -bit words) take unit time.

Given a starting vertex v , the distances to all other vertices in G can be computed in $O(m)$ time using breadth first search (BFS). The BFS tree rooted at v also gives shortest paths from v to all vertices. It is quite straightforward to also count for each edge in the BFS tree (starting from edges furthest from the root and moving towards the root) in how many shortest paths (starting from v) it participates. The counting requires only $O(n)$ operations, as there are only $n - 1$ edges in the BFS tree.

In general, several different BFS trees can be constructed starting at the same vertex v , because a vertex at distance i from v may have more than one neighbor at distance $i - 1$ from v . We shall need to construct two such trees. For both trees, we may scan the vertices of the graph in the following order, starting at v : within a level of the BFS tree, vertices are scanned in the order under which they were first discovered, and every vertex scans its neighbors in order of increasing labels. The *forward tree* rooted at v (which gives the lexicographically first shortest paths when v is the first vertex of the path) is constructed using the following rule: for every vertex discovered at level i keep a pointer to its level $i - 1$ neighbor that was first to be discovered (according to the scanning order described above). The *backward tree* rooted at v (which gives the lexicographically first shortest paths when v is the last vertex of the path) is constructed using the following rule: for every vertex discovered at level i keep a pointer to its level $i - 1$ neighbor of smallest label. Both the forward tree and the backward tree can be constructed in $O(m)$ time.

Given both the forward tree and the backward tree for a vertex v , and using the convention that for vertices with a label smaller than v one uses the backward tree and for vertices with a label larger than v one uses the forward tree, one can simultaneously count in $O(n)$ time how many shortest paths with an endpoint at v pass through every edge. (Note that this count is 0 for all but at most $2n - 2$ edges of the two BFS trees.) \square

We now consider m different graphs, one for every edge e . We denote the graph that we associate with edge e by G_e . The vertices of G_e are the vertices of G . Two vertices are connected by an edge in G_e iff e is on their unique shortest path in G . It follows that edge e is on c shortest paths in G iff the average degree in G_e is at least $2c/n$. Hence to find all weakest links, it suffices to find (or estimate) the average degrees of all graphs G_e . We shall now combine two facts.

1. By Theorem 4, $O(\sqrt{n/d_0}/\epsilon)$ degree queries suffice in order to estimate the average degree in a graph with average degree at least d_0 . To make the prob-

ability of error in this estimation below $1/n^2$, one can repeat the estimation procedure $O(\log n)$ times and take the median of the estimations. We shall set $d_0 = (1 - \epsilon)c/n$.

2. For any vertex v , Proposition 25 implies that in time $O(m)$ one can simultaneously obtain the degree of v in all graphs G_e .

Hence using $k = O\left(\frac{\log n \sqrt{n/(c/n)}}{\epsilon}\right) = O\left(\frac{n \log n}{\epsilon \sqrt{c}}\right)$ single source shortest path computations one can with high probability simultaneously estimate the average degree in all graphs G_e , and thus find all weakest links (edges that are on more than c shortest paths) while avoiding any false alarms (by edges that are on less than $(1/2 - \epsilon)c$ shortest paths). This proves Theorem 5.

Theorem 5 offers a savings of roughly n/\sqrt{c} in the running time compared to the running time of $O\left(\frac{mn^2 \log n}{c\epsilon^2}\right)$ in [3]. (Note, however, that ϵ has different meanings in the two bounds. Hence the savings comes at the cost of allowing more false alarms.)

Acknowledgments. I thank Johan Hastad, Michael Langberg, Eran Ofek, Gideon Schechtman, Benjy Weiss, and Avi Wigderson for their interest and involvement in various stages of this work, Assaf Naor for bringing to my attention the work of Samuels, and the anonymous referees for their detailed and useful comments.

REFERENCES

- [1] N. ALON AND J. SPENCER, *The Probabilistic Method*, Wiley-Interscience, New York, 2000.
- [2] Z. W. BIRNBAUM, J. RAYMOND, AND H. S. ZUCKERMAN, *A generalization of Tshebyshev's inequality to two dimensions*, Ann. Math. Statist., 18 (1947), pp. 70–79.
- [3] N. DEVANUR, R. LIPTON, AND N. VISHNOI, *Who's the weakest link?*, in Stochastic Algorithms: Foundations and Applications, Lecture Notes in Comput. Sci. 2827, A. Albrecht and K. Steinhöfel, eds., Springer, Berlin, 2003, pp. 108–116.
- [4] L. DUBINS AND L. SAVAGE, *Inequalities for Stochastic Processes (How to Gamble if You Must)*, Dover, New York, 1976.
- [5] O. GOLDRICH AND D. RON, *On Estimating the Average Degree of a Graph*, Electronic Colloquium on Computational Complexity (ECCC), Report TR04-13, 2004.
- [6] S. M. SAMUELS, *On a Chebyshev-type inequality for sums of independent random variables*, Ann. Math. Statist., 37 (1966), pp. 248–259.
- [7] S. M. SAMUELS, *The Markov inequality for sums of independent random variables*, Ann. Math. Statist. 40 (1969), pp. 1980–1984.
- [8] A. SIEGEL, *Median bounds and their application*, J. Algorithms, 38 (2001), pp. 184–236.

HIT-AND-RUN FROM A CORNER*

LÁSZLÓ LOVÁSZ† AND SANTOSH VEMPALA‡

Abstract. We show that the hit-and-run random walk mixes rapidly starting from any interior point of a convex body. This is the first random walk known to have this property. In contrast, the ball walk can take exponentially many steps from some starting points. The proof extends to sampling an exponential density over a convex body.

Key words. sampling, random walks, isoperimetric inequalities

AMS subject classifications. 52A20, 60J10, 68W20

DOI. 10.1137/S009753970544727X

1. Introduction. Consider a random walk in \mathbb{R}^n . It starts somewhere and at each step moves to a randomly chosen “neighboring” point (which could be the current point). With a suitable choice of the “neighbor” transition, the steady state distribution of such a walk can be the uniform distribution over a convex body or, indeed, any reasonable distribution in \mathbb{R}^n . For example, to sample uniformly from a convex body K , the *ball walk* at a point x chooses a point y uniformly in a ball of fixed radius centered at x and then goes to y if y is in K ; else, the step is wasted and it stays at x .

In the last decade and a half, there has been much progress in analyzing these walks [1, 2, 4, 6, 8, 9, 11, 12]. In [8] it was shown that the ball walk mixes in $O^*(n^3)$ steps from a *warm* start after appropriate preprocessing. (A warm start means that the starting point is chosen from a distribution that already is not too far from the target in the sense that its density at any point is at most twice the density of the target distribution. The O^* notation suppresses logarithmic factors and dependence on other parameters like error bounds.) While this result is sufficient to get polynomial-time algorithms for important applications, it is rather cumbersome to generate a warm start and increases the complexity substantially. Kannan and Lovász [6] have shown that the ball walk mixes in $O^*(n^3)$ time from any starting point if wasted steps are not counted. However, the ball walk can take an exponential number of (mostly wasted) steps to mix from some starting points, e.g., a point close to the apex of a rotational cone. (This is because most of the volume of the ball around the start is outside the cone.) Moreover, even starting from a fairly deep point (i.e., the distance to the boundary is much larger than the ball radius), the mixing time can be exponential.¹ The only known way to avoid this problem is to invoke a warm start; it has been an open question as to whether there is a random walk that mixes rapidly starting from, say, the center of gravity of the convex body.

*Received by the editors November 15, 2004; accepted for publication (in revised form) August 26, 2005; published electronically February 21, 2006.

<http://www.siam.org/journals/sicomp/35-4/44727.html>

†Microsoft Research, One Microsoft Way, Redmond, WA 98052 (lovasz@microsoft.com).

‡Department of Mathematics, Massachusetts Institute of Technology, Cambridge, MA 02139 (vempala@math.mit.edu).

¹Random walks on a discrete subset of \mathbb{R}^n (e.g., the lattice walk) avoid this local conductance problem but have other complications that make their convergence less efficient, although still polynomial. Also, one is sampling from a discrete subset, which might be acceptable for applications but is a bit unsatisfactory.

Is there a random walk that mixes rapidly starting from a(ny) single point? *Hit-and-run*, introduced by Smith [15], is defined as follows:

- Pick a uniformly distributed random line ℓ through the current point.
- Move to a uniform random point along the chord $\ell \cap K$.

It was proved in [15] that the stationary distribution of the hit-and-run walk is the uniform distribution π_K over K . In [10], it was shown that hit-and-run mixes in $O^*(n^3)$ steps from a warm start after appropriate preprocessing; i.e., it is no worse than the ball walk. In this paper, we show that it actually mixes rapidly from *any* interior starting point.

To be more precise, the mixing time *can* be big if we start from a very tight corner. But our bound will be logarithmic in the distance; thus, if, e.g., the convex body is described by a system of linear inequalities with rational coefficients, and the starting point is given by rational coordinates, then the mixing time will be polynomial in the input data.

To derive this mixing result, we prove the following theorem that still assumes a bound on the density of the starting distribution.

THEOREM 1.1. *Let K be a convex body that contains a ball of radius r and is contained in a ball of radius R . Let σ be a starting distribution and let σ^m be the distribution of the current point after m steps of hit-and-run in K . Let $\varepsilon > 0$, and suppose that the density function $d\sigma/d\pi_K$ is bounded by M except on a set S with $\sigma(S) \leq \varepsilon/2$. Then for*

$$m > 10^{10} \frac{n^2 R^2}{r^2} \ln \frac{M}{\varepsilon},$$

the total variation distance of σ^m and π_K is less than ε .

The condition on the starting density captures the case when the L_2 distance of σ and π is bounded (as shown in section 5). The theorem improves on existing bounds by reducing the dependence on M and ε from polynomial (which is unavoidable for the ball walk) to logarithmic, while maintaining the optimal dependence on r , R , and n . To bound the convergence to stationarity when starting from a specific point at distance d from the boundary, we do one step and then (if this is not too short) we apply Theorem 1.1 with the starting distribution obtained this way.

COROLLARY 1.2. *Under the conditions of Theorem 1.1, suppose that the starting distribution σ is concentrated on a single point in K at distance d from the boundary. Then for*

$$m > 10^{11} \frac{n^3 R^2}{r^2} \ln \frac{R}{d\varepsilon},$$

the total variation distance of σ^m and π_K is less than ε .

At the heart of this theorem is a bound of $\Omega(r/nR)$ on the conductance of every subset (see Theorem 4.2). (For the ball walk, the conductance of small sets can be arbitrarily small; therefore the need for a warm start.) As we discuss in section 5, the condition that K is contained in a ball of radius R can be replaced by the weaker condition that its second moment is at most R^2 , i.e., $\mathbf{E}_K(|x - z_K|^2) \leq R^2$, where z_K is the centroid of K . The mixing time goes up by a factor of $O(\ln^2(M/\varepsilon))$. For a body in near-isotropic position, $R/r = O(\sqrt{n})$ and so the number of steps required is $O(n^3 \ln^3(M/\varepsilon))$. It follows that hit-and-run mixes in $O(n^4 \ln^3(n/d))$ steps starting from a point at distance d from the boundary. Such a guarantee is not possible for the ball walk.

Our main tool is a new isoperimetric inequality (section 2). To formulate an isoperimetric inequality, one considers a partition of a convex body K into three sets S_1, S_2, S_3 such that S_1 and S_2 are “far” from each other and the inequality bounds the minimum possible volume of S_3 relative to the volumes of S_1 and S_2 . All previous inequalities have viewed the distance between S_1 and S_2 as the *minimum* distance between points in S_1 and points in S_2 . For example, if $d(S_1, S_2)$ is the minimum Euclidean distance between S_1 and S_2 , then

$$\text{vol}(S_3) \geq \frac{2d(S_1, S_2)}{D} \min\{\text{vol}(S_1), \text{vol}(S_2)\},$$

where D is the diameter of K [3, 7]. One can get a similar inequality using the cross-ratio distance (see section 2) instead of the Euclidean distance [10]:

$$\text{vol}(S_3) \geq d_K(S_1, S_2) \frac{\text{vol}(S_1)\text{vol}(S_2)}{\text{vol}(K)}.$$

In this paper, by means of a weight function $h(x)$ on K that measures the distance between S_1 and S_2 as a certain average distance, we obtain a more general inequality that can be much stronger. We formulate it for general logconcave functions in Theorem 2.1. For a convex body, it says that

$$\text{vol}(S_3) \geq \mathbf{E}_K(h(x)) \min\{\text{vol}(S_1), \text{vol}(S_2)\}.$$

The weight $h(x)$ at a point x is restricted only by the cross-ratio distance between pairs u, v from S_1, S_2 , respectively, for which $x \in [u, v]$. In general, the weight $h(x)$ can be much higher than the minimum cross-ratio distance between S_1 and S_2 .

Hit-and-run can be extended to sampling general densities f in \mathbb{R}^n as follows:

- Pick a uniformly distributed random line ℓ through the current point.
- Move to a random point y along the line ℓ chosen from the distribution induced by f on ℓ .

The stationary distribution of this walk is π_f , the probability measure with density f . It has been shown that it is efficient for any logconcave density from a warm start [13]. (Similar results are also known for the ball walk with a Metropolis filter [13].) It is natural to ask if hit-and-run is rapidly mixing from any starting point even for arbitrary logconcave functions. There are some technical problems with extending the results of this paper to arbitrary logconcave functions; but we make some progress in this direction by showing that this is indeed the case for an exponential density over a convex body. This class of density functions is interesting for other reasons as well—these are the functions used in “simulated annealing” and in the fastest volume algorithm [14]. We prove the following theorem in section 6. The condition on the starting density captures the case of bounded L_2 -norm; the proof uses the same isoperimetric inequality (see Theorem 2.1).

THEOREM 1.3. *Let $K \subseteq \mathbb{R}^n$ be a convex body and let f be a density supported on K which is proportional to $e^{-a^T x}$ for some vector $a \in \mathbb{R}^n$. Assume that the level set of f of probability $1/8$ contains a ball of radius r and that $\mathbf{E}_f(|x - z_f|^2) \leq R^2$, where z_f is the centroid of f . Let σ be a starting distribution and let σ^m be the distribution of the current point after m steps of hit-and-run applied to f . Let $\varepsilon > 0$, and suppose that the density function $d\sigma/d\pi_f$ is bounded by M except on a set S with $\sigma(S) \leq \varepsilon/2$. Then for*

$$m > 10^{30} \frac{n^2 R^2}{r^2} \ln^5 \frac{MnR}{r\varepsilon},$$

the total variation distance of σ^m and π_f is less than ε .

1.1. Overview of analysis. We wish to bound the rate of convergence of the Markov chain underlying hit-and-run to the uniform distribution π_K on the convex body K . For this we use the notion of *conductance*, which is defined as follows: For any measurable subset $S \subseteq K$ and $x \in K$, we denote by $P_x(S)$ the probability that a step from x goes to S . If $0 < \pi_K(S) < 1$, then the conductance $\phi(S)$ is defined as

$$\phi(S) = \frac{\int_{x \in S} P_x(K \setminus S) d\pi_K}{\min\{\pi_K(S), \pi_K(K \setminus S)\}}.$$

The minimum value over all subsets S is the conductance, ϕ , of the Markov chain. Lovász and Simonovits [12], extending a result of Jerrum and Sinclair [5], have shown that the mixing rate (roughly, the number of steps needed to halve the distance to the stationary distribution) is bounded by $O(1/\phi^2)$ (and is at least $1/\phi$).

The main part of our proof shows that the conductance of the hit-and-run Markov chain is $\Omega(r/nR)$. All previous attempts to bound the conductance of geometric random walks could prove only that the conductance of “large” subsets is large, namely, that the conductance bound for a subset S was proportional to $\pi_K(S)$. For this reason, one had to limit the probability that we start in one of bad small sets, which leads to the use of a warm start. As mentioned earlier, the example of starting at a point x near the apex of a rotational cone shows that the ball walk can in fact take exponentially many steps from some starting points: Most points of a ball around x are outside the cone, and hence most steps from x are wasted.

Hit-and-run, on the contrary, exhibits a sizable (inverse polynomial) drift toward the base of the cone. Thus, although the initial steps are tiny, they quickly get larger and the current point moves away from the apex. By bounding the conductance, we show that this phenomenon is general; i.e., hit-and-run mixes rapidly starting from any interior point of a convex body. To prove this, we use the new isoperimetric inequality. Besides the inequality, a key observation in the proof is that the “median” step length from points in K is a concave function.

2. A weighted isoperimetric inequality. To analyze the walk, we use a non-Euclidean notion of distance [10]. Let u, v be two distinct points in K , let $\ell(u, v)$ denote the line through u and v , and let p, q be the endpoints of the segment $\ell(u, v) \cap K$, so that the points appear in the order p, u, v, q along $\ell(u, v)$. Then,

$$d_K(u, v) = \frac{|u - v||p - q|}{|p - u||v - q|}.$$

For two subsets S_1, S_2 of K , we define

$$d_K(S_1, S_2) = \min_{u \in S_1, v \in S_2} d_K(u, v).$$

Theorem 2.5 from [13] asserts the following: If f is a logconcave function on a convex set K , $\varepsilon > 0$ and $S_1 \cup S_2 \cup S_3$ is a partition of K into three measurable sets such that for any $u \in S_1$ and $v \in S_2$ we have $d_K(u, v) \geq \varepsilon$, then

$$(1) \quad \int_K f(x) dx \int_{S_3} f(x) dx \geq \varepsilon \int_{S_1} f(x) dx \int_{S_2} f(x) dx.$$

We prove the following related result.

THEOREM 2.1. *Let K be a convex body in \mathbb{R}^n . Let $f : K \rightarrow \mathbb{R}_+$ be a logconcave function and let $h : K \rightarrow \mathbb{R}_+$ be an arbitrary function. Let S_1, S_2, S_3 be any partition*

of K into measurable sets. Suppose that for any pair of points $u \in S_1$ and $v \in S_2$ and any point x on the chord of K through u and v ,

$$h(x) \leq \frac{1}{3} \min(1, d_K(u, v)).$$

Then

$$\frac{\int_{S_3} f(x) dx}{\min \left\{ \int_{S_1} f(x) dx, \int_{S_2} f(x) dx \right\}} \geq \frac{\int_K h(x)f(x) dx}{\int_K f(x) dx}.$$

Remark. For a logconcave density function f and corresponding distribution π_f , the conclusion of the theorem can be stated as

$$\pi_f(S_3) \geq E_f(h(x)) \min\{\pi_f(S_1), \pi_f(S_2)\}.$$

Proof. We can assume that $\int_{S_1} f(x) dx \leq \int_{S_2} f(x) dx$. Suppose that the conclusion is false. Then there exists an $A \leq 1/2$ such that

$$\int_K f(x) dx = \frac{1}{A} \int_{S_1} f(x) dx$$

and

$$\int_{S_3} f(x) dx < A \int_K h(x)f(x) dx.$$

Now we invoke the localization lemma, specifically the version given in Corollary 2.4 of [7]. This implies that there exist two points $a, b \in K$ and a linear function $\ell : [0, 1] \rightarrow \mathbb{R}_+$ with the following properties. Set

$$F(t) = \ell(t)^{n-1} f(ta + (1-t)b), \quad G(t) = h(ta + (1-t)b),$$

and

$$J_i = \{t \in [0, 1] : ta + (1-t)b \in S_i\} \quad (i = 1, 2, 3);$$

then

$$\begin{aligned} \int_0^1 F(t) dt &= \frac{1}{A} \int_{J_1} F(t) dt, \\ \int_{J_3} F(t) dt &< A \int_0^1 G(t)F(t) dt, \end{aligned}$$

and hence

$$\int_0^1 F(t) dt \int_{J_3} F(t) dt < \int_{J_1} F(t) dt \int_0^1 G(t)F(t) dt.$$

For $u, v \in K$, let M_{uv} denote the maximum of $h(x)$ over the chord through u and v ; then

$$\int_0^1 G(t)F(t) dt \leq M_{ab} \int_0^1 F(t) dt,$$

and so

$$(2) \quad \int_{J_3} F(t) dt < M_{ab} \int_{J_1} F(t) dt.$$

We also have

$$(3) \quad \int_{J_1} F(t) dt = A \int_0^1 F(t) dt \leq \frac{1}{2} \int_0^1 F(t) dt.$$

Let $u \in J_1$ and $v \in J_2$, and (say) $u < v$. Then by hypothesis,

$$\frac{v - u}{u(1 - v)} \geq d_K(ua + (1 - u)b, va + (1 - v)b) \geq 3M_{ab},$$

and hence by the one-dimensional case of (1), we have

$$\int_0^1 F(t) dt \int_{J_3} F(t) dt \geq 3M_{ab} \int_{J_1} F(t) dt \int_{J_2} F(t) dt.$$

Comparing this with (2), we get

$$\int_0^1 F(t) dt > 3 \int_{J_2} F(t) dt.$$

Using this and (3), it follows that

$$\begin{aligned} \int_{J_3} F(t) dt &= \int_0^1 F(t) dt - \int_{J_1} F(t) dt - \int_{J_2} F(t) dt \\ &> \left(1 - \frac{1}{2} - \frac{1}{3}\right) \int_0^1 F(t) dt \\ &= \frac{1}{6} \int_0^1 F(t) dt. \end{aligned}$$

But then (2) and (3) imply that $M_{ab} > 1/3$, a contradiction. \square

3. Bounding the step size. For $x \in K$, let y be a random step from x . Following [10], we define $F(x)$ as

$$(4) \quad \mathbb{P}(|x - y| \leq F(x)) = \frac{1}{8}.$$

Roughly speaking, this is the “median” step length from x . The goal of this section is to bound this function from below by a concave function.

For $x \in K$, let

$$\lambda(x, t) = \frac{\text{vol}(K \cap (x + tB))}{\text{vol}(tB)}$$

denote the fraction of a ball of radius t around x that intersects K . For a fixed $\gamma \geq 0$, define $s : K \rightarrow \mathbb{R}_+$ as

$$s(x) = \sup\{t \in \mathbb{R}_+ : \lambda(x, t) \geq \gamma\}.$$

The value $s(x)$ is a measure of how close x is to the boundary of K . Its somewhat complicated definition guarantees some useful properties.

LEMMA 3.1. *For any $\gamma > 0$, $s(x)$ is a concave function.*

Proof. Let $x_1, x_2 \in K$ with $s(x_1) = r_1$ and $s(x_2) = r_2$. Let $A_i = K \cap (x_i + r_i B)$ ($i = 1, 2$). Let $x = (x_1 + x_2)/2$ and consider $A = (A_1 + A_2)/2$. By convexity, $A \subseteq K$. Further, any point $y \in A$ can be written as

$$y = \frac{1}{2}(x_1 + z_1 + x_2 + z_2) = x + \frac{z_1 + z_2}{2}$$

for some z_1, z_2 such that $|z_1| \leq r_1$ and $|z_2| \leq r_2$. Thus,

$$A \subseteq K \cap \left(x + \frac{r_1 + r_2}{2} B \right).$$

Next, by the Brunn–Minkowski inequality,

$$\begin{aligned} \text{vol}(A)^{\frac{1}{n}} &\geq \frac{1}{2} \left(\text{vol}(A_1)^{\frac{1}{n}} + \text{vol}(A_2)^{\frac{1}{n}} \right) \\ &\geq \frac{1}{2} \gamma^{\frac{1}{n}} \text{vol}(B)^{\frac{1}{n}} (r_1 + r_2) \\ &= \gamma^{\frac{1}{n}} \text{vol} \left(\frac{r_1 + r_2}{2} B \right)^{\frac{1}{n}}. \end{aligned}$$

It follows that

$$\text{vol} \left(K \cap \left(x + \frac{r_1 + r_2}{2} B \right) \right) \geq \text{vol}(A) \geq \gamma \text{vol} \left(\frac{r_1 + r_2}{2} B \right)$$

and thus $s(x) \geq (r_1 + r_2)/2$. \square

LEMMA 3.2. *If $\gamma \geq 63/64$, then for all $x \in K$,*

$$F(x) \geq \frac{s(x)}{32}.$$

Proof. Set $s = s(x)$. Let p denote the fraction of the surface of the ball $x + (s/2)B$ that is not in K . Then

$$\text{vol}((x + sB) \setminus K) \geq p \text{vol}(sB) - \text{vol}((s/2)B).$$

By the definition of s ,

$$\text{vol}((x + sB) \setminus K) \leq (1 - \gamma) \text{vol}(sB),$$

and hence

$$p \leq 1 - \gamma + 2^{-n} \leq \frac{1}{32}.$$

Take a random line ℓ through x ; then with probability at least $1 - 2p$, $\ell \cap (x + (s/2)B) \subseteq K$. If this happens, then for the point y chosen uniformly from $\ell \cap K$, we have

$$\mathbb{P} \left(|y - x| \leq \frac{s}{32} \mid \ell \right) \leq \frac{1}{16},$$

and so

$$P\left(|y - x| \leq \frac{s}{32}\right) \leq \frac{1}{16} + \frac{15}{16} \cdot \frac{1}{16} < \frac{1}{8}.$$

This proves the lemma. \square

Let us quote Corollary 4.6 in [8] as the following lemma.

LEMMA 3.3. *Suppose K contains a ball of radius r . Then,*

$$\int_K \int_{y \in x+tB \setminus K} dy dx \leq \frac{t\sqrt{n}}{2r} \text{vol}(K) \text{vol}(tB).$$

This lemma can be used to bound the average value of $s(x)$ from below, as follows.

LEMMA 3.4. *Suppose K contains a unit ball. Then,*

$$\int_K s(x) dx \geq \frac{1-\gamma}{\sqrt{n}} \text{vol}(K).$$

Proof. From Lemma 3.3,

$$\int_K \lambda(x, t) dx \geq \left(1 - \frac{t\sqrt{n}}{2}\right) \text{vol}(K).$$

On the other hand,

$$\int_K \lambda(x, t) dx \leq \gamma \text{vol}(K) + (1-\gamma) \text{vol}(\{x \in K : \lambda(x, t) \geq \gamma\})$$

and so

$$\text{vol}(\{x \in K : \lambda(x, t) \geq \gamma\}) \geq \left(1 - \frac{t\sqrt{n}}{2(1-\gamma)}\right) \text{vol}(K).$$

Using this,

$$\begin{aligned} \int_K s(x) dx &= \int_0^\infty \text{vol}(\{x \in K : \lambda(x, t) \geq \gamma\}) dt \\ &\geq \text{vol}(K) \int_0^\infty \left(1 - \frac{t\sqrt{n}}{2(1-\gamma)}\right)^+ dt \\ &= \frac{1-\gamma}{\sqrt{n}} \text{vol}(K). \quad \square \end{aligned}$$

4. A scale-free bound on the conductance. For a point $u \in K$, let P_u be the distribution obtained by taking one hit-and-run step from u . Then (as shown in [10]),

$$(5) \quad P_u(A) = \frac{2}{\text{vol}_{n-1}(\partial B)} \int_A \frac{dx}{\ell(u, x)|x - u|^{n-1}},$$

where $\ell(u, x)$ is the length of the chord through u and x .

Let $d_{tv}(P, Q)$ denote the total variation distance between distributions P and Q . The following lemma from [10] connects the geometric distance of two points to the variation distance of the distributions obtained by taking one hit-and-run step.

LEMMA 4.1 (see [10]). *Let $u, v \in K$. Suppose that*

$$d_K(u, v) < \frac{1}{8} \text{ and } |u - v| < \frac{2}{\sqrt{n}} \max\{F(u), F(v)\}.$$

Then

$$d_{tv}(P_u, P_v) < 1 - \frac{1}{500}.$$

The main theorem of this section is the following.

THEOREM 4.2. *Let K be a convex body in \mathbb{R}^n of diameter D , containing a unit ball. Then the conductance of hit-and-run in K is at least $\frac{1}{2^{24}nD}$.*

Proof. Let $K = S_1 \cup S_2$ be a partition into measurable sets. We will prove that

$$(6) \quad \int_{S_1} P_x(S_2) dx \geq \frac{1}{2^{24}nD} \min\{\text{vol}(S_1), \text{vol}(S_2)\}.$$

We can read the left-hand side as follows: We select a random point X from the uniform distribution and make one step to get Y . What is the probability that $X \in S_1$ and $Y \in S_2$? It is well known that this quantity remains the same if S_1 and S_2 are interchanged.

Consider the points that are deep inside these sets, i.e., unlikely to jump out of the set:

$$S'_1 = \left\{ x \in S_1 : P_x(S_2) < \frac{1}{1000} \right\}$$

and

$$S'_2 = \left\{ x \in S_2 : P_x(S_1) < \frac{1}{1000} \right\}.$$

Let S'_3 be the rest, i.e., $S'_3 = K \setminus S'_1 \setminus S'_2$.

Suppose $\text{vol}(S'_1) < \text{vol}(S_1)/2$. Then

$$\int_{S_1} P_x(S_2) dx \geq \frac{1}{1000} \text{vol}(S_1 \setminus S'_1) \geq \frac{1}{2000} \text{vol}(S_1),$$

which proves (6).

So we can assume that $\text{vol}(S'_1) \geq \text{vol}(S_1)/2$ and, similarly, that $\text{vol}(S'_2) \geq \text{vol}(S_2)/2$. For any $u \in S'_1$ and $v \in S'_2$,

$$d_{tv}(P_u, P_v) \geq 1 - P_u(S_2) - P_v(S_1) > 1 - \frac{1}{500}.$$

Thus, by Lemma 4.1, either

$$(7) \quad d_K(u, v) \geq \frac{1}{8}$$

or

$$(8) \quad |u - v| \geq \frac{2}{\sqrt{n}} \max\{F(u), F(v)\}.$$

We want to apply Theorem 2.1 to the partition S'_1, S'_2, S'_3 and the function $h(x) = s(x)/(48D\sqrt{n})$, where $s(x)$ is as defined in section 3 with $\gamma = 63/64$. To verify the condition, let $u \in S'_1, v \in S'_2$, and x be any point on the chord pq through u and v (where p is the endpoint closer to u than v). Clearly $h(x) \leq 1/3$. If (7) holds, then $h(x) \leq d_K(u, v)/3$ is trivial. Then suppose that (8) holds. Let, e.g., x be between u and q . Then, using the concavity of s (Lemma 3.1), we have

$$\begin{aligned} s(x) &\leq \frac{|x-p|}{|u-p|}s(u) \leq 32\frac{|q-p|}{|u-p|}F(u) \quad (\text{using Lemma 3.2}) \\ &\leq 16\frac{|q-p|}{|u-p|}\sqrt{n}|u-v| \quad (\text{using (8) above}) \\ &= 16d_K(u, v)\sqrt{n}|q-v| \\ &\leq 16d_K(u, v)D\sqrt{n}, \end{aligned}$$

and hence $h(x) \leq d_K(u, v)/3$ follows again. Thus, Theorem 2.1 applies with f being the uniform density and we get

$$\begin{aligned} \frac{\text{vol}(S'_3)}{\min\{\text{vol}(S'_1), \text{vol}(S'_2)\}} &\geq \frac{1}{48D\sqrt{n}} \cdot \frac{1}{\text{vol}(K)} \int_K s(x) dx \\ &> \frac{1}{4000nD}. \end{aligned}$$

Here we have used Lemma 3.4 with $\gamma = 63/64$. Therefore,

$$\begin{aligned} \int_{S_1} P_x(S_2) dx &\geq \frac{1}{2} \cdot \frac{1}{1000} \text{vol}(S'_3) \\ &\geq \frac{1}{2^{23}nD} \min\{\text{vol}(S'_1), \text{vol}(S'_2)\} \\ &\geq \frac{1}{2^{24}nD} \min\{\text{vol}(S_1), \text{vol}(S_2)\}, \end{aligned}$$

which again proves (6). \square

5. Proof of the mixing bound. First note that hit-and-run is invariant under a scaling of space (i.e., there is a 1-1 mapping between the random walk in K and cK) and thus the conductance bound of $\Omega(r/nR)$ follows by considering K/r . Next, suppose we start with an M -warm distribution σ ; i.e., for any subset S of K , $\sigma(S) \leq M\pi_K(S)$. Then using Corollary 1.5 of [12], the distribution σ^m obtained after m steps satisfies

$$d_{tv}(\sigma^m, \pi_K) \leq \sqrt{M} \left(1 - \frac{\phi^2}{2}\right)^m$$

and thus after $m > Cn^2 \frac{R^2}{r^2} \ln \frac{M}{\epsilon}$ steps (C is a constant), the total variation distance of σ^m and π_K is less than ϵ .

If we know only that $\sigma \leq M\pi_K$ except for the subsets of a set S with $\sigma(S) < \epsilon/2$, then we think of a random point of K as being generated with probability $1 - \epsilon/2$ from a distribution σ' that is $(2M)$ -warm with respect to π_K and with probability $\epsilon/2$ from some other distribution. After m steps, we have

$$d_{tv}(\sigma^m, \pi_K) \leq \frac{\epsilon}{2} + \left(1 - \frac{\epsilon}{2}\right) \sqrt{\frac{2M}{\epsilon}} \left(1 - \frac{\phi^2}{2}\right)^m,$$

which implies Theorem 1.1.

In some applications, the L_2 -norm of σ w.r.t. π is bounded; i.e., suppose that

$$\int_K \left(\frac{d\sigma}{d\pi}\right)^2 d\pi \leq M.$$

This will also be sufficient for mixing. The set

$$S = \left\{x : \frac{d\sigma}{d\pi} > \frac{2M}{\varepsilon}\right\}$$

has measure $\pi(S)$ of at most $\varepsilon/2$. So we can apply the mixing theorem with $2M/\varepsilon$ in place of M .

As mentioned in the introduction, Theorem 1.1 can be strengthened to require only that $\mathbb{E}_K(\|x - z_K\|^2) \leq R^2$ with a small increase in the mixing time. It is well known that the volume of K outside a ball of radius $R \ln(2/\delta)$ is at most a $\delta/2$ fraction. Thus the conductance of any subset of measure x is at least

$$\phi(x) = \frac{cr}{nR \ln(2/x)}$$

for some constant c . Then the average conductance theorem of [6] implies that after $m > C(n^2 R^2 / r^2) \ln^3(M/\varepsilon)$ steps (where C is a constant), we get that $d_{tv}(\sigma^m, \pi_K) \leq \varepsilon$.

Finally, Corollary 1.2 follows by bounding M for the distribution obtained after one step of hit-and-run.

6. Exponential density over a convex body. Here we extend the main theorem to sample an exponential density over a convex body. We will use the following notation. Let f be a density function in \mathbb{R}^n . For any line ℓ in \mathbb{R}^n , let $\mu_{\ell, f}$ be the measure induced by f on ℓ , i.e.,

$$\mu_{\ell, f}(S) = \int_{p+tu \in S} f(p+tu) dt,$$

where p is any point on ℓ and u is a unit vector parallel to ℓ . We abbreviate $\mu_{\ell, f}$ by μ_ℓ if f is understood, and also $\mu_\ell(\ell)$ by μ_ℓ . The probability measure $\pi_\ell(S) = \mu_\ell(S)/\mu_\ell$ is the restriction of f to ℓ .

For two points $u, v \in \mathbb{R}^n$, let $\ell(u, v)$ denote the line through them. Let $[u, v]$ denote the segment connecting u and v , and let $\ell^+(u, v)$ denote the semiline in ℓ starting at u and not containing v . Furthermore, let

$$\begin{aligned} f^+(u, v) &= \mu_{\ell, f}(\ell^+(u, v)), \\ f^-(u, v) &= \mu_{\ell, f}(\ell^+(v, u)), \\ f(u, v) &= \mu_{\ell, f}([u, v]). \end{aligned}$$

For any $T > 0$, let $L(T) = \{x : f(x) \geq T\}$ be the level set of function value T . It will be convenient to let π_n denote the volume of the unit ball in \mathbb{R}^n .

6.1. Distance. The following “distance” was used in [13]:

$$d_f(u, v) = \frac{f(u, v)f(\ell(u, v))}{f^-(u, v)f^+(u, v)}.$$

(This quantity is not really a distance, since it does not satisfy the triangle inequality. To get a proper distance function, one could consider $\ln(1 + d_f(u, v))$; but it will be more convenient to work with d_f .)

Note that when f is the uniform distribution over a convex set K , then $d_f(u, v) = d_K(u, v)$. The next lemma describes how the two are related in general.

LEMMA 6.1. *Let f be a logconcave density function in \mathbb{R}^n whose support is a convex body K . Let $G = \max_K f(x)/\min_K f(x)$.*

1. $d_f(u, v) \geq d_K(u, v)$.
- 2.

$$d_K(u, v) \geq \frac{\min\{3, d_f(u, v)\}}{6(1 + \ln G)}.$$

The first inequality is Lemma 5.9 in [13], and the second inequality is a direct implication of Lemma 5.11 in [13].

6.2. Step size. Let f be a density function whose support is a convex body K . We define three parameters that all measure the local smoothness of f . First, for a fixed β and γ , we define

$$\lambda(x, t) = \frac{\text{vol}((x + tB) \cap L(\beta f(x)))}{\text{vol}(tB)} \quad \text{and} \quad s(x) = \sup\{t \in \mathbb{R}_+ : \lambda(x, t) \geq \gamma\}.$$

Second, we define $F(x)$ by

$$\mathbb{P}(|x - y| \leq F(x)) = \frac{1}{8},$$

where y is a random step from x . Third, we define $\alpha(x)$ (as in [13]) as the smallest $s \geq 3$ for which a hit-and-run step y from x satisfies

$$\mathbb{P}(f(y) \geq sf(x)) \leq \frac{1}{16}.$$

We will shortly fix $\beta = 3/4$ and $\gamma = 63/64$. Note that $\lambda(x, t)$, $s(x)$, and $F(x)$ as defined here are generalizations of the definitions in section 3 (where $f(x)$ was the uniform density over K).

The following lemma was proved in [13].

LEMMA 6.2 (see [13, Lemma 6.10]).

$$\pi_f(u : \alpha(u) \geq t) \leq \frac{16}{t}.$$

Our next lemma extends a crucial property of $s(x)$ to exponential functions (it does not hold for general logconcave functions).

LEMMA 6.3. *Suppose $f(x)$ is proportional to $e^{-a^T x}$ in a convex body K and zero outside. Then for any fixed $\beta, \gamma > 0$, the function $s(x)$ is concave.*

Proof. Let $x_1, x_2 \in K$ with $s(x_1) = r_1$ and $s(x_2) = r_2$. Define

$$A_1 = \{y \in x_1 + r_1B : f(y) \geq \beta f(x_1)\} \quad \text{and} \quad A_2 = \{y \in x_2 + r_2B : f(y) \geq \beta f(x_2)\}.$$

Now let $x = (x_1 + x_2)/2$ and consider $A = (A_1 + A_2)/2$. Any point $y \in A$ can be written as

$$y = x + \frac{z_1 + z_2}{2}$$

for some z_1, z_2 such that $z_1 \in r_1 B$ and $z_2 \in r_2 B$. Thus

$$A \subseteq x + \frac{r_1 + r_2}{2} B.$$

Also, since $f(x)$ is proportional to $e^{-a^T x}$, we have $f((x + y)/2) = \sqrt{f(x)f(y)}$ and so for any $y \in A$,

$$f(y) = f\left(\frac{y_1 + y_2}{2}\right) = \sqrt{f(y_1)f(y_2)},$$

where $y_1 \in A_1$ and $y_2 \in A_2$. By the definition of these subsets, $f(y_1) \geq \beta f(x_1)$ and $f(y_2) \geq \beta f(x_2)$. Thus

$$f(y) \geq \beta \sqrt{f(x_1)f(x_2)} = \beta f\left(\frac{x_1 + x_2}{2}\right) = \beta f(x)$$

and so

$$A \subseteq \left\{y \in x + \frac{r_1 + r_2}{2} B : f(y) \geq \beta f(x)\right\}.$$

Finally, by the Brunn–Minkowski inequality,

$$\begin{aligned} \text{vol}(A)^{\frac{1}{n}} &\geq \frac{1}{2} \left(\text{vol}(A_1)^{\frac{1}{n}} + \text{vol}(A_2)^{\frac{1}{n}}\right) \\ &\geq \frac{1}{2} (\gamma \pi_n)^{\frac{1}{n}} (r_1 + r_2) \\ &= \gamma^{\frac{1}{n}} \text{vol}\left(\frac{r_1 + r_2}{2} B\right)^{\frac{1}{n}}. \end{aligned}$$

It follows that $s(x) \geq (r_1 + r_2)/2$. \square

Next, we bound the expected value of $s(x)$.

LEMMA 6.4. *Let f be any logconcave density such that the level set of f of measure $1/8$ contains a ball of radius r . Then with $\beta = 3/4$ and $\gamma = 63/64$,*

$$\mathbb{E}_f(s(x)) \geq \frac{r}{2^{10} \sqrt{n}}.$$

Proof. Let L_0 be the level set

$$L_0 = \{x : f(x) \geq f_0\},$$

such that the measure of L_0 is $1/8$. For $i = 1, 2, \dots$, consider the level sets

$$L_i = \left\{x : f(x) \geq \left(\frac{3}{4}\right)^i f_0\right\}.$$

Note that since f is logconcave, each L_i is a convex body. We will first bound $\mathbb{E}_f(1 - \lambda(x, t))$ as follows:

$$\begin{aligned}
 \int_{\mathbb{R}^n} f(x) \int_{y \in x+tB: f(y) < 3f(x)/4} \frac{dy}{\text{vol}(tB)} dx & \\
 & \leq \frac{1}{8} + \sum_{i>0} \frac{f_0}{(4/3)^{i-1}} \int_{x \in L_i \setminus L_{i-1}} \int_{y \in x+tB \setminus L_i} \frac{dy}{\text{vol}(tB)} dx \\
 & \leq \frac{1}{8} + \sum_i \frac{f_0}{(4/3)^{i-1}} \int_{x \in L_i} \int_{y \in x+tB \setminus L_i} \frac{dy}{\text{vol}(tB)} dx \\
 & \leq \frac{1}{8} + \frac{t\sqrt{n}}{2r} \sum_i \frac{f_0}{(4/3)^{i-1}} \text{vol}(L_i).
 \end{aligned}$$

In the last step, we applied Lemma 3.3 to the convex set L_i which contains a ball of radius r by assumption. Now for any $x \in L_i \setminus L_{i-1}$,

$$\frac{f_0}{(4/3)^i} \leq f(x) < \frac{f_0}{(4/3)^{i-1}}.$$

Using this,

$$\begin{aligned}
 \sum_i \frac{f_0}{(4/3)^{i-1}} \text{vol}(L_i) & \leq \sum_i \frac{4f_0}{(4/3)^{i-1}} \text{vol}(L_i \setminus L_{i-1}) \\
 & \leq \frac{16}{3} \int_{\mathbb{R}^n} f(x) dx < 6.
 \end{aligned}$$

Thus,

$$\mathbb{E}_f(1 - \lambda(x, t)) \leq \frac{1}{8} + \frac{3t\sqrt{n}}{r}.$$

Next, since $\lambda(x, t)$ can be at most 1, we get

$$\int_{x: \lambda(x, t) \geq 3/4} f(x) dx \geq \frac{1}{2} - \frac{12t\sqrt{n}}{r}.$$

We will use the following claim to complete the proof: If $\lambda(x, t) \geq 3/4$, then for $c > 1$,

$$\lambda(x, t/c) \geq 1 - e^{-(\frac{c}{4}-1)^2/2}.$$

To see the claim, note that since $\lambda(x, t) \geq 3/4$, there must be a ball of radius $t/2\sqrt{n}$ inside K centered at x . The claim then follows by applying Lemma 4.4 in [13].

Setting $c = 16$ above, we get $\lambda(x, t/16) \geq 1 - e^{-9/2} > 63/64$. Using this,

$$\begin{aligned}
 \int_{\mathbb{R}^n} s(x)f(x) dx & \geq \frac{1}{16} \int_{t=0}^\infty \int_{x: \lambda(x, t) \geq 3/4} f(x) dx dt \\
 & \geq \frac{1}{16} \int_{t=0}^\infty \left(\frac{1}{2} - \frac{12t\sqrt{n}}{r} \right)^+ dt \\
 & \geq \frac{r}{2^{10}\sqrt{n}}. \quad \square
 \end{aligned}$$

We can also relate the maximum value of $s(x)$ to the diameter D .

LEMMA 6.5. Let $G = \frac{\max_K f(x)}{\min_K f(x)}$, where f is proportional to $e^{-a^T x}$ with support K . Suppose K has diameter D . Then,

$$\max_K s(x) \leq \min \left\{ \frac{2\sqrt{n}D}{\ln G}, D \right\}.$$

Proof. Let $t = 1/|a|$. Then along the direction of a , the function value drops by $1/e$ each time we move distance t . Hence,

$$t \leq \frac{D}{\ln G}.$$

On the other hand, for any point x , we claim that

$$s(x) \leq 2t\sqrt{n}.$$

To see this, consider the nearest point y along the line through x in the direction of a with $f(y) \leq f(x)/2$. This point satisfies $|x - y| \leq t$. Now the portion of the ball $x + s(x)B$ in the half-space $\{z : a^T z \geq a^T y\}$ must have volume at most $1/4$ of the volume of $s(x)B$ by the definition of $s(x)$ (in a ball of radius $2t\sqrt{n}$, a half-space at distance t from the center cuts off at least $1/4$ of the volume of the ball). This implies the inequality. The lemma follows. \square

The next lemma is about the step size along a given line.

LEMMA 6.6. Let f be logconcave and ℓ be any line through a point x . Let p, q be intersection points of ℓ with the boundary of $L(F/8)$, where F is the maximum value of f along ℓ , and let $s = \max\{|x - p|/32, |x - q|/32\}$. Choose a random point y on ℓ from the distribution π_ℓ . Then

$$P(|x - y| > s) > \frac{3}{4}.$$

Proof. We will use the following observation. For any logconcave function g that is nonincreasing on an interval $[a, b]$

$$\int_{[a,b]} g(x) dx \geq |a - b| \frac{g(a) - g(b)}{\ln g(a) - \ln g(b)}.$$

The proof is by noting that the exponential function with value $g(a)$ at a and $g(b)$ at b is a lower bound on any such function.

In our case, suppose f attains its maximum at a point $z \in [p, q]$. Then, applying the observation separately to the intervals $[p, z]$ and $[z, q]$, we get

$$\int_{[p,q]} f(x) dx \geq \frac{7F}{8 \ln 8} |p - q|.$$

Also, by Lemma 3.5(a) in [13] (whose proof uses a similar reduction to the exponential function), $P(y \in [p, q]) \geq 7/8$. We now consider two cases. If $x \in [p, q]$, then $s \leq |p - q|/32$ and thus

$$P(|x - y| \leq s) \leq \frac{2sF}{\int_{[p,q]} f(x) dx} \leq \frac{\ln 8}{14} < \frac{1}{4}.$$

Suppose $x \notin [p, q]$. Let u be the unit vector along $p - q$. Then,

$$|[x - su, x + su] \cap [p, q]| \leq \frac{|p - q|}{32},$$

and thus

$$\mathbb{P}(|x - y| \leq s) \leq \frac{1}{8} + \frac{F|p - q|/32}{7F|p - q|/8 \ln 8} < \frac{1}{4}. \quad \square$$

Finally, $s(x)$ gives a lower bound on $F(x)$ as in section 3.

LEMMA 6.7. *If $\gamma \geq 63/64$ and $\beta \geq 3/4$, then*

$$F(x) \geq \frac{s(x)}{64}.$$

Proof. We need to prove the following: If $x \in \mathbb{R}^n$ and $s > 0$ satisfies

$$\text{vol}((x + sB) \cap \{f \leq \beta f(x)\}) \leq (1 - \gamma)\text{vol}(x + sB),$$

then for a hit-and-run step y from x ,

$$(9) \quad \mathbb{P}\left(|x - y| \leq \frac{s}{64}\right) \leq \frac{1}{8}.$$

Let p denote the fraction of the surface of $x + (s/2)B$ in the set $\{f \leq \beta f(x)\}$. Clearly

$$\begin{aligned} \text{vol}((x + sB) \cap \{f \leq \beta f(x)\}) &\geq p\text{vol}(x + sB) - \text{vol}(x + (s/2)B) \\ &= (p - 2^{-n})\text{vol}(x + sB), \end{aligned}$$

and by our hypothesis on s ,

$$p \leq 1 - \gamma + 2^{-n} \leq \frac{1}{32}.$$

Thus if we choose a random line through x , with probability at least $15/16$ it will intersect the surface of $x + (s/2)B$ in points z_1, z_2 with $f(z_i) \geq \beta f(x)$.

Suppose that we have chosen such a line, and let u be a unit vector parallel to this line. Then we have

$$f(x + tu) \geq \beta f(x) \quad \left(-\frac{s}{2} \leq t \leq \frac{s}{2}\right)$$

and also (by logconcavity)

$$f(x + tu) \leq \beta^{-2|t|/s} f(x) \quad (-\infty < t < \infty).$$

We have

$$\mathbb{P}(|x - y| \leq s/64) = \int_{-s/64}^{s/64} f(x + tu) dt \Big/ \int_{-\infty}^{\infty} f(x + tu) dt.$$

Here

$$\int_{-\infty}^{\infty} f(x + tu) dt \geq \int_{-s/2}^{s/2} f(x + tu) dt \geq s\beta f(x),$$

while

$$\int_{-s/64}^{s/64} f(x + tu) dt \leq \frac{s}{32} \beta^{-1/32} f(x),$$

and thus

$$\mathbb{P}\left(|x - y| \leq \frac{s}{64}\right) \leq \frac{1}{32} \beta^{-33/32} < \frac{1}{16}.$$

Thus the probability that $|x - y| \leq s/64$ is bounded by $2p + 1/16 \leq 1/8$. This proves the lemma. \square

6.3. Conductance. For a point $u \in K$, let P_u be the distribution obtained by taking one hit-and-run step from u . Let $\mu_f(u, x)$ be the integral of f along the line through u and x . Then,

$$(10) \quad P_u(A) = \frac{2}{n\pi_n} \int_A \frac{f(x) dx}{\mu_f(u, x)|x - u|^{n-1}}.$$

The next lemma is analogous to Lemma 4.1. It holds for any logconcave density f , although we know how to use it only for the exponential density. Its proof is closely related to that of Lemma 7.2 in [13].

LEMMA 6.8. *Let $u, v \in K$. Suppose that*

$$d_f(u, v) < \frac{1}{128 \ln(3 + \alpha(u))} \quad \text{and} \quad |u - v| < \frac{1}{4\sqrt{n}} \max\{F(u), F(v)\}.$$

Then

$$d_{tv}(P_u, P_v) < 1 - \frac{1}{500}.$$

Proof. We will show that there exists a set $A \subseteq K$ such that $P_u(A) \geq \frac{1}{2}$ and for any subset $A' \subset A$,

$$P_v(A') \geq \frac{1}{200} P_u(A').$$

To this end, we define certain “bad” lines through u . Let σ be the uniform probability measure on lines through u .

Let B_1 be the set of lines that are not almost orthogonal to $u - v$, in the sense that for any point $x \neq u$ on the line,

$$|(x - u)^T(u - v)| > \frac{2}{\sqrt{n}}|x - u||u - v|.$$

The measure of this subset can be bounded as $\sigma(B_1) \leq 1/8$.

Next, let B_2 be the set of all lines through u which contain a point y with $f(y) > 2\alpha(u)f(u)$ (see section 6.2 for the definition of α). By Lemma 3.5(a) in [13], if we select a line from B_2 , then with probability at least $1/2$, a random step along this line takes us to a point x with $f(x) \geq \alpha(u)f(u)$. From the definition of $\alpha(u)$, this can happen with probability at most $1/16$, which implies that $\sigma(B_2) \leq 1/8$.

Let A be the set of points x in K which are not on any of the lines in $B_1 \cup B_2$, and which are far from u in the sense of Lemma 6.6:

$$|x - u| \geq \frac{1}{32} \max\{|u - p|, |u - q|\}.$$

Applying Lemma 6.6 to each such line, we get

$$P_u(A) \geq \left(1 - \frac{1}{8} - \frac{1}{8}\right) \frac{3}{4} > \frac{1}{2}.$$

We will show that for any subset $A' \subseteq A$,

$$P_v(A') \geq \frac{1}{200} P_u(A')$$

using the next two claims.

Claim 1. For every $x \in A$,

$$|x - v| \leq \left(1 + \frac{1}{n}\right) |x - u|.$$

Claim 2. For every $x \in A$,

$$\mu_f(v, x) < 64 \frac{|x - v|}{|x - u|} \mu_f(u, x).$$

Claim 1 is easy to prove (cf. [10]), and the proof of Claim 2 is identical to that given in [13]. Thus, for any $A' \subset A$,

$$\begin{aligned} P_v(A') &= \frac{2}{n\pi_n} \int_{A'} \frac{f(x) dx}{\mu_f(v, x) |x - v|^{n-1}} \\ &\geq \frac{2}{64n\pi_n} \int_{A'} \frac{|x - u| f(x) dx}{\mu_f(u, x) |x - v|^n} \\ &\geq \frac{2}{64en\pi_n} \int_{A'} \frac{f(x) dx}{\mu_f(u, x) |x - u|^{n-1}} \\ &\geq \frac{1}{64e} P_u(A'). \end{aligned}$$

The lemma follows. \square

We are now ready to state and prove the main theorem.

THEOREM 6.9. *Let f be a density in \mathbb{R}^n proportional to $e^{-\alpha^T x}$ whose support is a convex body K of diameter D . Assume that any level set of measure $1/8$ contains a ball of radius r . Then for any subset S , with $\pi_f(S) = p \leq 1/2$, the conductance of hit-and-run satisfies*

$$\phi(S) \geq \frac{r}{10^{13} n D \ln \frac{nD}{rp}}.$$

Proof. The proof has the same structure as that of Theorem 4.2.

Let $K = S_1 \cup S_2$ be a partition into measurable sets, where $S_1 = S$ and $p = \pi_f(S_1) \leq \pi_f(S_2)$. We will prove that

$$(11) \quad \int_{S_1} P_x(S_2) dx \geq \frac{r}{10^{13} n D \ln \frac{nD}{rp}} \pi_f(S_1).$$

Consider the points that are deep inside these sets:

$$S'_1 = \left\{ x \in S_1 : P_x(S_2) < \frac{1}{1000} \right\} \quad \text{and} \quad S'_2 = \left\{ x \in S_2 : P_x(S_1) < \frac{1}{1000} \right\}.$$

Let S'_3 be the rest, i.e., $S'_3 = K \setminus S'_1 \setminus S'_2$.

Suppose $\pi_f(S'_1) < \pi_f(S_1)/2$. Then

$$\int_{S_1} P_x(S_2) dx \geq \frac{1}{1000} \pi_f(S_1 \setminus S'_1) \geq \frac{1}{2000} \pi_f(S_1),$$

which proves (11).

So we can assume that $\pi_f(S'_1) \geq \pi_f(S_1)/2$ and, similarly, that $\pi_f(S'_2) \geq \pi_f(S_2)/2$.

Next, define the exceptional subset W as the set of points u for which $\alpha(u)$ is very large.

$$W = \left\{ u \in S : \alpha(u) \geq \frac{2^{27}nD}{rp} \right\}.$$

By Lemma 6.2,

$$\pi_f(W) \leq \frac{rp}{2^{23}nD}.$$

Next, for any $u \in S'_1 \setminus W$ and $v \in S'_2 \setminus W$,

$$d_{tv}(P_u, P_v) \geq 1 - P_u(S_2) - P_v(S_1) > 1 - \frac{1}{500}.$$

Thus, by Lemma 6.8, either

$$d_f(u, v) \geq \frac{1}{128 \ln(3 + \alpha(u))} \geq \frac{1}{2^{12} \ln \frac{nD}{rp}}$$

or

$$|u - v| \geq \frac{1}{4\sqrt{n}} \max\{F(u), F(v)\}.$$

But by Lemmas 6.1 and 6.7, this implies that either

$$(12) \quad d_K(u, v) \geq \frac{1}{2^{15} \ln \frac{nD}{rp} (1 + \ln G)}$$

or

$$(13) \quad |u - v| \geq \frac{1}{2^8 \sqrt{n}} \max\{s(u), s(v)\}$$

holds. Now, by Lemma 6.5, condition (12) implies that

$$(14) \quad d_K(u, v) \geq \frac{1}{2^{17} \ln \frac{nD}{rp}} \frac{\max s(x)}{\sqrt{n}D}.$$

Next, we define

$$h(x) = \frac{s(x)}{2^{19} D \sqrt{n} \ln \frac{nD}{rp}}$$

and apply Theorem 2.1 to the partition $S'_1 \setminus W$, $S'_2 \setminus W$ and the rest. If (14) holds, then clearly $h(x) \leq d_K(u, v)/3$. Otherwise, (13) holds. Let x be a point on the chord pq of K , say between u and q . Then, using the concavity of s (Lemma 6.3),

$$\begin{aligned} s(x) &\leq \frac{|x - p|}{|u - p|} s(u) \leq 2^8 \frac{|q - p|}{|u - p|} \sqrt{n} |u - v| \\ &\leq 2^8 d_K(u, v) D \sqrt{n} \end{aligned}$$

and hence $h(x) \leq d_K(u, v)/3$ again. Thus,

$$\begin{aligned} \pi_f(S'_3) &\geq \mathbb{E}_f(h)\pi_f(S'_1 \setminus W)\pi_f(S'_2 \setminus W) - \pi_f(W) \\ &\geq \frac{r}{2^{30}nD \ln \frac{nD}{ra}}\pi_f(S_1). \end{aligned}$$

Here we have used Lemma 6.4 and the bound on $\pi_f(W)$. Therefore,

$$\begin{aligned} \int_{S_1} P_x(S_2) dx &\geq \frac{1}{2} \cdot \frac{1}{1000}\pi_f(S'_3) \\ &\geq \frac{r}{10^{13}nD \ln \frac{nD}{rp}}\pi_f(S_1), \end{aligned}$$

which again proves (11). \square

6.4. Mixing time. Since f satisfies $\mathbb{E}_f(|x - z_f|^2) \leq R^2$, we consider the restriction of f to the ball of radius $R \ln(4e/a)$ around z_f and then by Lemma 5.17 in [13], the measure of f outside this ball is at most $a/4$. In the proof of the conductance bound, we can consider the restriction of f to this set. In the bound on the conductance for a set of measure a , the diameter D is effectively replaced by $R \ln(4e/a)$.

The bound on the mixing time then follows by applying Theorem 6.9 along with either Corollary 1.6 in [12] or the average conductance theorem of [6]. For the latter, we have that for any subset of measure x , the conductance is at least

$$\phi(x) \geq \frac{cr}{nR \ln(nR/rx) \ln(4e/x)} \geq \frac{cr}{nR \ln^2(nR/rx)}.$$

Then the theorem of [6] implies that after $m > C(n^2R^2/r^2) \ln^5(MnR/r\varepsilon)$ steps, we have $d_{tv}(\sigma^m, \pi_f) < \varepsilon$.

REFERENCES

- [1] D. APPELATE AND R. KANNAN, *Sampling and integration of near log-concave functions*, in Proceedings of the 23rd ACM Symposium on Theory of Computing (STOC), 1991, pp. 156–163.
- [2] R. BUBLEY, M. DYER, AND M. JERRUM, *An elementary analysis of a procedure for sampling points in a convex body*, Random Structures Algorithms, 12 (1998), pp. 213–235.
- [3] M. E. DYER AND A. M. FRIEZE, *Computing the volume of a convex body: A case where randomness provably helps*, in Proceedings of the AMS Symposium on Probabilistic Combinatorics and Its Applications, 1991, pp. 123–170.
- [4] M. DYER, A. FRIEZE, AND R. KANNAN, *A random polynomial-time algorithm for approximating the volume of convex bodies*, J. Assoc. Comput. Mach., 38 (1991), pp. 1–17.
- [5] M. JERRUM AND A. SINCLAIR, *Approximating the permanent*, SIAM J. Comput., 18 (1989), pp. 1149–1178.
- [6] R. KANNAN AND L. LOVÁSZ, *Faster mixing through average conductance*, in Proceedings of the 31st ACM Symposium on Theory of Computing (STOC), 1999, pp. 282–287.
- [7] R. KANNAN, L. LOVÁSZ, AND M. SIMONOVITS, *Isoperimetric problems for convex bodies and a localization lemma*, Discrete Comput. Geom., 13 (1995) pp. 541–559.
- [8] R. KANNAN, L. LOVÁSZ, AND M. SIMONOVITS, *Random walks and an $O^*(n^5)$ volume algorithm for convex bodies*, Random Structures Algorithms, 11 (1997), pp. 1–50.
- [9] L. LOVÁSZ, *How to compute the volume?*, in Jber. d. Dt. Math.-Verein, Jubiläumstagung 1990, Teubner, Stuttgart, Germany, 1992, pp. 138–151.
- [10] L. LOVÁSZ, *Hit-and-run mixes fast*, Math. Program 86 (1999), pp. 443–461.
- [11] L. LOVÁSZ AND M. SIMONOVITS, *Mixing rate of Markov chains, an isoperimetric inequality, and computing the volume*, in Proceedings of the 31st Annual IEEE Symposium on Foundations of Computer Science, 1990, pp. 482–491.

- [12] L. LOVÁSZ AND M. SIMONOVITS, *Random walks in a convex body and an improved volume algorithm*, Random Structures Algorithms, 4 (1993), pp. 359–412.
- [13] L. LOVÁSZ AND S. VEMPALA, *The geometry of logconcave functions and sampling algorithms*, in Proceedings of the 44th Symposium on Foundations of Computer Science (FOCS), 2003, pp. 640–649; available online at <http://www-math.mit.edu/~vempala/papers/logcon.pdf>.
- [14] L. LOVÁSZ AND S. VEMPALA, *Simulated annealing in convex bodies and an $O^*(n^4)$ volume algorithm*, J. Comput. System Sci. (special issue for FOCS 2003), to appear.
- [15] R.L. SMITH, *Efficient Monte-Carlo procedures for generating points uniformly distributed over bounded regions*, Oper. Res., 32 (1984), pp. 1296–1308.

FUNCTION MATCHING*

AMIHOOD AMIR[†], YONATAN AUMANN[†], MOSHE LEWENSTEIN[†], AND ELY PORAT[†]

Abstract. We present problems in the following three application areas: identifying similar codes in which global register reallocation and spill code minimization were done (programming languages); protein threading (computational biology); and searching for color icons under different color maps (image processing).

We introduce a new search model called *function matching* that enables us to solve the above problems. The *function matching problem* has as its input a text T of length n over alphabet Σ_T and a pattern $P = P[1]P[2] \cdots P[m]$ of length m over alphabet Σ_P . We seek all text locations i , where the m -length substring that starts at i is equal to $f(P[1])f(P[2]) \cdots f(P[m])$, for some function $f : \Sigma_P \rightarrow \Sigma_T$.

We give a randomized algorithm that solves the function matching problem in time $O(n \log n)$ with probability $\frac{1}{n}$ of declaring a false positive. We give a deterministic algorithm whose time is $O(n|\Sigma_P| \log m)$ and show that it is optimal in the convolutions model. We use function matching to efficiently solve the problem of two-dimensional parameterized matching.

Key words. pattern matching, parameterized matching, function matching, color maps

AMS subject classifications. 68Q05, 68Q20, 68Q25

DOI. 10.1137/S0097539702424496

1. Introduction. The last few decades have prompted the evolution of pattern matching from a combinatorial solution of the exact string matching problem [19, 20] to an area concerned with approximate matching of various relationships motivated by computational molecular biology, computer vision, and complex searches in digitized and distributed multimedia libraries [18, 9]. We will describe a number of important applications in various diverse areas that necessitate the introduction of a new generalized matching paradigm, that of *function matching*.

Programming languages. The *parameterized matching* problem was introduced by Baker [11]. Her main motivation lay in software maintenance, where program fragments are considered “identical” even if variable names are different. Therefore, strings under this model are comprised of symbols from two disjoint sets Σ and Π containing fixed symbols and parameter symbols, respectively. In this paradigm, one seeks *parameterized occurrences*, i.e., occurrences up to renaming of the parameter symbols, of one string in another. This renaming is a bijection $b : \Pi \rightarrow \Pi$.

Important topics in compiler design are global register allocation and spill code minimization (see, e.g., [23, 22]). The idea is to minimize the number of used registers by reusing the same register for several purposes.

Unfortunately, the parameterized matching paradigm will not identify similar codes with different levels of optimizing reusable global variables because of the bijection requirement on the parameter alphabet Π . In this application, it is sufficient that $f : \Pi \rightarrow \Pi$ is a function.

*Received by the editors December 3, 2002; accepted for publication (in revised form) July 18, 2005; published electronically March 3, 2006.

<http://www.siam.org/journals/sicomp/35-5/42449.html>

[†]Department of Computer Science, Bar-Ilan University, 52900 Ramat-Gan, Israel (amir@cs.biu.ac.il, aumann@cs.biu.ac.il, moshe@cs.biu.ac.il, porately@cs.biu.ac.il). The work of the first author was partially supported by ISF grant 282/01. Part of this work was done while the first author was at Georgia Tech, College of Computing, and was supported by NSF grant CCR-01-04494. The work of the third author was partially supported by an IBM Faculty Award.

Computational biology. The Grand Challenge *protein folding* problem is one of the most important problems in computational biology (see, e.g., [24]). It consists of determining the protein's tertiary structure from its linear arrangement of peptides.

This problem is still open, which leads to the myriad of extremely active research methods used in the attempt to solve it. One possible technique that is being investigated is *threading* (see, e.g., [12, 31]). The idea is to try to “thread” a given protein sequence into a known structure. A possible way of viewing this idea is to consider subsequences that are known to fold in a particular way. These sequences can be used as patterns in subsequent given sequences with unknown secondary or tertiary structure. However, a subsequence of different peptides that bond in the same way as the pattern peptides may still fold in a similar way. This cannot be detected by exact matching, yet a function matching (under a careful reduction) will detect such a match.

Image Processing. One of the interesting problems in Web searching is searching for color images (see, e.g., [27, 10, 4]). Assume, for example, that we are seeking a given icon in any possible color map. If the colors were fixed, then this is exact two-dimensional pattern matching [3]. However, if the color map is changed the exact matching algorithm would not find the pattern. A parameterized two-dimensional search is precisely what is needed. If, in addition, we are also willing to lose resolution, then we would use a two-dimensional function matching search.

The above examples are just a sample of diverse application areas encountering search problems that are not solved by state-of-the-art methods in pattern matching. This has led us to introduce, in this paper, the *function matching* model, and to explore the *two-dimensional parameterized matching* problem. We start with an intuitive description of the function matching model.

In the traditional pattern matching model, one seeks exact occurrences of a given pattern in a text, i.e., text locations where every text symbol is *equal* to its corresponding pattern symbol. In parameterized matching, one seeks occurrences of a pattern image achieved by a bijection on the alphabet symbols. We seek text locations where there exists a bijection f on the alphabet for which every text symbol is equal to $f(a)$, where a is the corresponding pattern symbol. In the applications we described, f cannot be a bijection. Rather, it should be just a function. P matches T at location i if, for every element $a \in \Pi$, all occurrences of a have the same corresponding symbol in T . In other words, unlike in parameterized matching, there may be *several* different symbols in the pattern which are mapped to the same text symbol.

Relaxing the bijection restriction introduces nontrivial technical difficulties. Many powerful pattern matching techniques such as automata methods [20, 13], subword trees [30, 14], dueling [28, 3], and deterministic sampling [29] assume transitivity of the matching relation. For any pattern matching relation where transitivity does not exist, the above methods do not help.

Examples of pattern matching with nontransitive matching relation are string matching with “don't cares” [19], less-than matching [6], pattern matching with mismatches [1, 21], and swapped matching [25, 2, 5]. It is interesting to note that the efficient algorithms for solving the above problems all use convolutions as their main tool. Convolutions were introduced by Fischer and Paterson [19] as a technique for solving pattern matching problems where the match relation is not transitive.

It turns out that many such problems can be solved by a “standard” application of convolutions (e.g., matching with “don't cares,” matching with mismatches in bounded finite alphabets, and swapped matching). Muthukrishnan and Palem were the first to identify this application method as the *convolutions model* [26]. The con-

volution model provides an excellent tool for solving many nonstandard matching problems efficiently using an off-the-shelf method. Even more important, a rigorous formal definition of such a model can be very useful in proving lower bounds. While such bounds do not restrict the solution complexity in a general RAM, they do help in understanding the limits of the convolution method, hitherto the only powerful tool for nonstandard pattern matching. Unfortunately, we have not found in the literature a formal definition of the convolutions model.

There are three main contributions in this paper as follows:

1. *A solution to a number of search problems in diverse fields.* This is achieved by the introduction of a new type of generalized pattern matching, called function matching.
2. *A formalization of the convolutions model.* This leads to a deterministic solution. We prove that this solution is *tight* in the convolutions model. We also present an efficient randomized solution of the function matching problem.
3. *A solution to the problem of exact search in color images with different color maps.* This is done via efficient randomized and deterministic algorithms for two-dimensional parameterized and function matching.

This paper is organized in the following way. In section 2 we give basic definitions. In sections 3 and 4 we present progressively more efficient deterministic solutions, culminating in an $O(n|\Sigma_P| \log m)$ algorithm, where $|\Sigma_P|$ is the pattern alphabet size. In section 5 we present a Monte Carlo algorithm that solves the function matching problem in time $O(n \log n)$ with failure probability no larger than $\frac{1}{n}$. In section 6 we formalize the convolutions model. We then show a lower bound proving that our deterministic algorithm is tight in the convolutions model and discuss the limitations of that model. Finally, in section 7 we present a randomized algorithm that solves the two-dimensional parameterized matching problem in time $O(n^2 \log n)$ with probability of false positives no larger than $\frac{1}{n^2}$. We also present a deterministic algorithm that solves the two-dimensional parameterized matching problem in time $O(n^2 m \log^2 m \log \log m)$.

2. Problem definition. Formally, *function pattern matching* is defined as follows.

DEFINITION. *Let s_1 and s_2 be arrays of the same dimensions and sizes over alphabets Σ_1 and Σ_2 , respectively. Array s_1 is said to function-match array s_2 if there exists a function $f : \Sigma_1 \rightarrow \Sigma_2$, such that $f(s_1) = s_2$, where $f(s_1)$ is the array obtained by replacing every occurrence of $x \in \Sigma_1$ with $f(x)$. (Note that this implies that such an f is always a surjection.) If f is a bijection, then s_1 is said to parameterize-match.*

Given two arrays $P = P[i, j]$, $i, j = 0, \dots, m - 1$, a pattern of size m^2 over alphabet Σ_P , and $T = T[i, j]$, $i, j = 0, \dots, n - 1$, a text of size n^2 over alphabet Σ_T , there is a *function occurrence* (*parameterized occurrence*, respectively) of P in T at location $[i, j]$ if P function-matches (*parameterize-matches*, respectively) the $m \times m$ subarray of T that begins at the $[i, j]$ th position ($T[i + k, j + \ell]$, $k, \ell = 0, \dots, m - 1$).

The problem in which the pattern and text are strings, i.e., one-dimensional arrays, is a special case of the above.

The problem of function matching (parameterized matching) is the following:

INPUT: Pattern P and text T .

OUTPUT: All function (parameterized, respectively) occurrences of P in T .

The parameterized matching definition above differs from Baker's original definition, since her motivation called for two types of alphabet. Our simpler definition

follows the definition of Amir, Farach, and Muthukrishnan, who also proved in [7] that the two definitions are mutually linearly reducible.

Observation 1 establishes the connection between function matching and parameterized matching.

OBSERVATION 1. *Let s_1 and s_2 be strings of the same length over alphabet Σ , and assume that string s_1 function-matches string s_2 . Then s_1 parameterize-matches s_2 iff $|\Sigma_1| = |\Sigma_2|$.*

Proof. By the definition of cardinality, two sets have the same cardinality iff there is a bijection between them. \square

3. Naive algorithm for function matching. The function matching problem can be trivially solved in time $O(nm)$ simply by checking if there exists a match for every location (for unbounded alphabets, there is also a $\log m$ multiplicative factor).

Convolutions can be used in a standard fashion to improve the time for finite fixed alphabets.

DEFINITION. *Let $P[0], \dots, P[m-1]$ and $T[0], \dots, T[n-1]$ be arrays of natural numbers. The discrete convolution (polynomial multiplication) of T and P is R , where*

$$R[j] = \sum_{i=0}^{m-1} T[j-i]P[i],$$

where $j = 0, \dots, n-m$. We denote R as $T * P$.

The convolution can be computed in time $O(n \log m)$, in a computational model with word size $O(\log m)$, by using the fast Fourier transform (FFT) [17].

Similarly to R , we can define the following array M :

$$M[j] = \sum_{i=0}^{m-1} T[j+i]P[i],$$

where $j = 0, \dots, n-m+1$. We denote M as $T \otimes P$. It is easy to see that computing R and M can easily reduce them to each other by reversing P .

Notations. For $\sigma \in \Sigma$, let

$$\chi_\sigma(x) = \begin{cases} 1 & \text{if } x = \sigma, \\ 0 & \text{if } x \neq \sigma. \end{cases}$$

If $X = x_1, \dots, x_n$ then $\chi_\sigma(X) = \chi_\sigma(x_1), \dots, \chi_\sigma(x_n)$.

Let $a \in \Sigma_T$, $b \in \Sigma_P$. Then $\chi_a(T) \otimes \chi_b(P)[i]$ is precisely the number of times that an a in the text matches a b in the pattern when the pattern is matched starting at text location i . Formally, $\chi_a(T) \otimes \chi_b(P)[i] = |\{\ell \in \{0, \dots, m-1\} \mid T[i+\ell] = a \text{ and } P[\ell] = b\}|$. If that number is equal to the number of b 's in the pattern, then we know that in text location i all pattern b 's are mapped to a in the text.

This leads to the following algorithm for the function matching problem.

ALGORITHM A.
 For all $b \in \Sigma_P$ do
 $c_b \leftarrow$ the number of b 's in P
 For all $a \in \Sigma_T$, $b \in \Sigma_P$ do
 $F_{a,b} \leftarrow (\chi_a(T) \otimes \chi_b(P))$
 endfor
 endfor
 Announce a function occurrence of P at every location i for which $\forall b \in \Sigma_P \exists a \in \Sigma_T$ such that $F_{a,b}[i] = c_b$. END

Time. There are $|\Sigma_T||\Sigma_P|$ convolutions and each convolution requires $O(n \log m)$ steps. Hence the total time is $O(n|\Sigma_T||\Sigma_P| \log m)$.

For fixed finite alphabets this algorithm is quite efficient. For unbounded alphabets, the time may be $O(n^2 m \log m)$, which is significantly worse than the naive algorithm.

4. Deterministic improvements. In this section we reduce the $|\Sigma_T|$ factor to $\log |\Sigma_T|$. We use a standard observation allowing text size reduction and use an interesting lemma that allows text alphabet size reduction for the function matching problem.

The following observation is commonly used in pattern matching papers (see, e.g., [6, 2]).

OBSERVATION 2. *To assume that the text size is $n \leq 2m$ does not limit the algorithm's generality. The reason is that if a matching problem can be deterministically solved in time $O(f(m))$ for text of size $n \leq 2m$, then it can be solved in time $O(\frac{n}{m} f(m))$ for any n -length strip. Simply divide the text into $2\frac{n}{2m}$ overlapping $2m$ -length segments and solve the matching problem separately for each. Clearly, all locations are covered.*

From this point on our paper assumes that the text size $n \leq 2m$.

DEFINITION. A binary alphabet reduction function is a function $f : \Sigma \rightarrow \{0, 1\}$.

LEMMA 1. *Let P and T be a pattern and text over alphabets Σ_P and Σ_T , respectively. Then there exist $\log |\Sigma_T|$ binary alphabet reduction functions $\{f_1, \dots, f_{\log |\Sigma_T|}\}$ such that, for every text location i of T , there is a function occurrence of P iff, for all $f \in \{f_1, \dots, f_{\log |\Sigma_T|}\}$, there is a function occurrence of P at location i of $f(T)$.*

Proof. For $a \in \Sigma_T$, let $f_i(a)$ be the i th bit of a 's binary representation, $i = 1, \dots, \log |\Sigma_T|$.

(\Rightarrow) This follows immediately from the function matching definition.

(\Leftarrow) If there is no function occurrence of P at location i of T , it means that there exist two indices j and k , $j, k \in \{0, \dots, m-1\}$, for which $P[j] = P[k]$ but $T[i+j] \neq T[i+k]$. However, $T[i+j] \neq T[i+k]$ means that there is some bit in their binary representation, where $T[i+j]$ and $T[i+k]$ are different. Let that bit be the ℓ th bit. Then $f_\ell(T[i+j]) \neq f_\ell(T[i+k])$, and therefore there is no function occurrence of P at location i of $f_\ell(T)$. \square

COROLLARY 1. *The function matching problem can be solved in time $O(n|\Sigma_P| \log^2 m)$.*

Proof. Construct the $O(\log |\Sigma_T|)$ binary alphabet reduction functions and use Algorithm A to solve the function matching problem of P in $f_\ell(T)$ for all $\ell = 1, \dots, \log |\Sigma_T|$. Each run of Algorithm A takes time $O(n|\Sigma_P| \log m)$, and there are $\log |\Sigma_T|$ applications of Algorithm A. Since the text length is $n \leq 2m$, it follows that $|\Sigma_T| \leq 2m$, and hence $\log |\Sigma_T| = O(\log m)$ bits. Therefore the total time is $O(n|\Sigma_P| \log^2 m)$. \square

The following number theoretic lemma allows for the removal of yet another logarithmic factor. The lemma is folklore. Nevertheless, we include a proof for the sake of completeness.

LEMMA 2. *Let a_1, \dots, a_k be natural numbers. Then $k \sum_{i=1}^k (a_i)^2 = (\sum_{i=1}^k a_i)^2$ iff $a_i = a_j$, $i, j = 1, \dots, k$.*

Proof. If $a_i = a_j$, $i, j = 1, \dots, k$ then it is easily seen that the equations are equal. Conversely, we show a stronger claim, $k \sum_{i=1}^k (a_i)^2 \geq (\sum_{i=1}^k a_i)^2$, and show that $k \sum_{i=1}^k (a_i)^2 > (\sum_{i=1}^k a_i)^2$ iff there exists i such that $a_i \neq a_k$. The proof is by induction.

The base case is easily verified. Assume that $k \sum_{i=1}^k (a_i)^2 \geq (\sum_{i=1}^k a_i)^2$ and

prove for $k + 1$. We have $(k + 1) \sum_{i=1}^{k+1} (a_i)^2 = k \sum_{i=1}^k (a_i)^2 + \sum_{i=1}^k (a_i)^2 + (k + 1)(a_{k+1})^2$ and $(\sum_{i=1}^{k+1} a_i)^2 = (\sum_{i=1}^k a_i + a_{k+1})^2 = (\sum_{i=1}^k a_i)^2 + a_{k+1}^2 + 2a_{k+1}(\sum_{i=1}^k a_i)$. By induction, $k \sum_{i=1}^k (a_i)^2 \geq (\sum_{i=1}^k a_i)^2$, and hence it is sufficient to show that $\sum_{i=1}^k (a_i)^2 + k(a_{k+1})^2 \geq 2a_{k+1}(\sum_{i=1}^k a_i)$, which is equivalent to $\sum_{i=1}^k a_i(a_i - a_{k+1}) - \sum_{i=1}^k a_{k+1}(a_i - a_{k+1}) \geq 0$ and can be shortened to $\sum_{i=1}^k (a_i - a_{k+1})^2 \geq 0$, which is obviously true. Moreover, it is easily seen that $\sum_{i=1}^k (a_i - a_{k+1})^2 > 0$ only if there is an i such that $a_i \neq a_{k+1}$. \square

For every $a \in \Sigma_P$ and for every location i in the text, we are interested in finding out if there is indeed a single symbol that appears in every text location that matches a in the occurrence beginning at text location i . Lemma 2 allows us to do that with a constant number of convolutions.

The convolutions are defined as follows: Assume that $\Sigma_T = \{1, \dots, |\Sigma_T|\}$. If the alphabet is not of this form, it can be turned into $\{1, \dots, |\Sigma_T|\}$ by sorting the alphabet and renaming the characters. This can always be accomplished in time $O(m \log m)$. Let $k = |\Sigma_P|$. Let T_2 be the array of length n , where $T_2[i] = (T[i])^2$, $i = 0, \dots, n - 1$, where $(T[i])^2$ is the square of the number $T[i]$.

The locations i , where $k(T_2 \otimes \chi_a(P))[i] = (T \otimes (\chi_a(P)))[i]^2$, are precisely the locations where all a 's are matched to a single text symbol.

COROLLARY 2. *The function matching problem can be solved in time $O(n|\Sigma_P| \log m)$.*

Proof. Because of Lemma 2, our algorithm requires only $O(|\Sigma_P|)$ convolutions. \square

5. Randomized improvements. In this section we present a randomized Monte Carlo algorithm that solves the function matching problem in time $O(n \log n)$. Our probability of declaring a function occurrence when it does not exist is $\frac{1}{n}$.

We have shown reductions that minimized the text alphabet size as well as the text size. Next we attempt to reduce the number of *occurrences* of an alphabet symbol in the pattern. Note that a symbol that appears only once in the pattern does not have any effect on the function matching, since it can be matched to any text symbol without ever causing a contradiction. In particular, a pattern that consists of m different symbols function-matches every text location. However, the following lemma shows that the crucial number of character occurrences is 2.

DEFINITION. *A pattern is called a paired pattern if no symbol in the pattern appears more than twice.*

LEMMA 3. *Let P and T be text and pattern over Σ_P and Σ_T , respectively. Then there exist paired patterns P_{even} and P_{odd} such that there is a function occurrence of P in location i of T iff there are function occurrences of both P_{even} and P_{odd} in location i of T .*

Proof. For every $a \in \Sigma$, let c_a be the number of times a appears in the pattern, and let those appearances be in locations $\ell_0^a, \dots, \ell_{c_a-1}^a$. Let $s_0^a, \dots, s_{\lfloor c_a/2 \rfloor}^a$ be new symbols. P_{even} is constructed from P by replacing $P[\ell_i^a]$ with $s_{\lfloor i/2 \rfloor}^a$. P_{odd} is constructed from P by replacing $P[\ell_i^a]$ with $s_{\lfloor i/2 \rfloor}^a$.

In P_{even} we have replaced the first pair of occurrences of every letter with a new symbol, the second pair by a new symbol, and so on. In P_{odd} we shift the starting point of the pairs by 1. The first occurrence of a symbol is replaced by a new symbol, each symbol of the next pair is replaced by the next new symbol, and so on.

Example. Let $P = a a a b a b a b b a a b b$. Then

$$P_{even} = s_0^a s_0^a s_1^a s_0^b s_1^a s_0^b s_2^a s_1^b s_1^b s_2^a s_3^a s_2^b s_2^b,$$

$$P_{odd} = s_0^a s_1^a s_1^a s_0^b s_2^a s_1^b s_2^a s_1^b s_2^b s_3^a s_3^a s_2^b s_3^b.$$

(\Rightarrow) If there is a function occurrence of P in location i of T , then breaking up symbol occurrences into pairs cannot make matters worse and there will be function occurrences of both P_{even} and P_{odd} in location i of T .

(\Leftarrow) If there is no function occurrence of P in location i of T , then there must be a pair ℓ_j^a, ℓ_{j+1}^a where the corresponding symbols do not match, i.e., $T[i+\ell_j^a] \neq T[i+\ell_{j+1}^a]$. This pair will cause a function mismatch either at P_{even} , if j is even, or at P_{odd} , if j is odd. \square

Lemma 3 implies that if we find an efficient algorithm for solving the function matching problem for paired patterns, we are done. Our randomized algorithm will do just that. The idea behind our algorithm is to make sure that every pair is matched with the same symbol. We will measure it by assigning the text and pattern numerical values and by assigning a positive number to the first element of a pattern pair and the negation of that number to the second element of a pattern pair. Thus if a pair of pattern symbols $x, -x$ matches the same text symbol y , then $xy + -xy = 0$. We need to discuss the conditions under which the probability of a false positive is small.

ALGORITHM B.
 Let $f_T : \Sigma_T \rightarrow \{1, \dots, n^2\}$, where $f_T(\sigma)$ is chosen uniformly at random.
 Let $g_P : \Sigma_P \rightarrow \{1, \dots, n^2\}$, where $g_P(\sigma)$ is chosen uniformly at random, with the exception that any symbol appearing only once in the pattern is always mapped to 0.
 Let $f_P : \Sigma_P \times \{\text{first}, \text{second}\} \rightarrow \{-n^2, \dots, n^2\}$, where $f_P(\sigma, \text{first}) = g_P(\sigma)$ and $f_P(\sigma, \text{second}) = -g_P(\sigma)$.
 $M \leftarrow f_T(T) \otimes f_P(P)$
 For $i = 0$ to $n - 1$:
 If $M[i] = 0$, then declare function matching in location i
 endfor
 END

Time. The algorithm performs one convolution, so its time is $O(n \log m)$ word operations for words of size $O(\log n)$ bits. If we stay within the $O(\log m)$ bit-word computation model, the time is $O(n \log n)$.

We need to show that our algorithm gives the correct answer with high probability.

THEOREM 1. *If there is a function occurrence of P in location i of T , then Algorithm B will announce it. Conversely, the probability that Algorithm B will incorrectly announce a function occurrence at some location is $\frac{1}{n}$.*

Proof. If there is a function occurrence of P in location i of T , then for every pair in P , the corresponding text symbols are equal. Consider a pair in locations j and k of the pattern $f_P(P[j]) = x, f_P(P[k]) = -x, f_T(t[i+j]) = y$, and $f_T(t[i+k]) = y$. The contribution of this pair to a convolution result at location i is thus $xy - xy = 0$. Because there is a function occurrence, this is true for every pair. The single pattern elements were mapped to 0. Therefore the value of the convolution in location i is 0.

Conversely, if Algorithm B wrongly announces a function occurrence at position i , it is because the convolution result at location i was 0. Consider a pair j, k , where the corresponding text elements are not equal, i.e., $P[j] = P[k] = a$ and $T[i+j] = b$ and $T[i+k] = c$.

By definition, $Pr(f_T(T) \otimes f_P(P)[i] = 0) = Pr(\sum_{l=1}^m f_T(T[i+l]) \cdot f_P(P[l]) = 0)$.

Hence,

$$\begin{aligned} & Pr(f_T(T) \otimes f_P(P)[i] = 0) \\ = & Pr\left(\sum_{l=1}^m f_T(T[i+l]) \cdot f_P(P[l]) - (f_T(T[i+j]) \cdot f_P(P[j]) + f_T(T[i+k]) \cdot f_P(P[k]))\right) \\ = & (f_T(T[i+j]) \cdot f_P(P[j]) + f_T(T[i+k]) \cdot f_P(P[k])) \end{aligned}$$

Denote with p_0 the following:

$$\begin{aligned} & \sum_{l=1}^m f_T(T[i+l]) \cdot f_P(P[l]) - (f_T(b) \cdot f_P(a, first) + f_T(c) \cdot f_P(a, second)) \\ & = \sum_{l=1}^m f_T(T[i+l]) \cdot f_P(P[l]) - (g_P(a) \cdot [f_T(b) - f_T(c)]). \end{aligned}$$

We will show that $Pr((g_P(a) \cdot [f_T(b) - f_T(c)]) = p_0) \leq \frac{1}{n^2}$.

Note that $f_T(b)$ and $f_T(c)$ were chosen randomly from $\{1, \dots, n^2\}$ and are independent of each other. Hence, for any constant z_0 , $Pr([f_T(b) - f_T(c)] = z_0) \leq \frac{1}{n^2}$. For any $z_0 \neq 0$, $Pr(g_P(a) \cdot z_0) \leq \frac{1}{n^2}$ since $g_P(a)$ was chosen randomly from $\{1, \dots, n^2\}$. Moreover, for $z_0 = 0$, $g_P(a) \cdot z_0 = 0$, and for $z_0 \neq 0$, $g_P(a) \cdot z_0 \neq 0$. Hence, for $z_0 = 0$, $Pr(g_P(a) \cdot z_0) \leq \frac{1}{n^2}$. Since $P[h] \neq a$, for each $h \notin \{j, k\}$, it follows that $Pr((g_P(a) \cdot [f_T(b) - f_T(c)]) = p_0) \leq \frac{1}{n^2}$, i.e., $Pr(f_T(T) \otimes f_P(P)[i] = 0) \leq \frac{1}{n^2}$.

It follows that over all n locations, $Pr(\exists i, f_T(T) \otimes f_P(P)[i] = 0) \leq \frac{1}{n}$. \square

6. Lower bounds in the convolutions model. In this section we give compelling evidence that an efficient deterministic solution to the function matching problem, if such a solution exists, may be difficult. We do so by showing an $\Omega(nm)$ lower bound for the problem in the *convolutions model*. This model, which we formally define shortly, captures the modus operandi of pattern matching algorithms that use convolutions as their key computational tool. We note that this model is different from the convolution model of [26], which is defined in combinatorial terms, unlike our computational model.

6.1. The convolutions model. We begin by defining the type of problems that are solved by the convolutions model. For a string T , denote by T_i the suffix of T starting at position i and by T_i^j the substring from T from location i to j inclusive.

DEFINITION 1. A pattern matching problem is defined as follows:

- *Match relation:* a binary relation $M(a, b)$, where $a \in \Sigma_P^*$ and $b \in \Sigma_T^*$;
- *Input:* a pattern $P = P[0], \dots, P[m - 1]$ and a text $T = T[0], \dots, T[n - 1]$, $P[i] \in \Sigma_P, T[j] \in \Sigma_T$.
- *Output:* the set of indices $S \subseteq \{0, \dots, n - 1\}$, where the pattern P matches to T according to M , i.e., all indices i such that $M(P, T_i)$ (where T_i is the suffix of T starting at location i).

As the name suggests, the convolutions model uses convolutions as the key operation for computing the output for a pattern matching problem. In most cases the convolutions are not performed directly between the original text and the original pattern, but rather between different *transformations* thereof. Thus, the first step is to compute a sequence of transformation $T^{(1)}, \dots, T^{(c)}$ for the text and $P^{(1)}, \dots, P^{(c)}$ for the pattern. Then, for each k , one computes the *convolution* $C^{(k)} = T^{(k)} \otimes P^{(k)}$.

Given these convolutions, for each location i the decision on whether there is a match at i is determined based on the values $C^{(1)}[i], \dots, C^{(c)}[i]$.

A computation in the convolution model is thus composed of the following three phases:

1. Preprocessing: producing the transformations $T^{(1)}, \dots, T^{(c)}$ and $P^{(1)}, \dots, P^{(c)}$.
2. Convolutions: computing $C^{(k)} = T^{(k)} \otimes P^{(k)}$ for $k = 1, \dots, c$.
3. Postprocessing: for each $i = 0, \dots, n - 1$ deciding if there is a match at location i based on $C^{(1)}[i], \dots, C^{(c)}[i]$.

The convolution phase is the key computational tool in the convolution model on the total number of steps of the entire process. Thus, we must determine what operations are permitted in the pre- and postprocessing phases (or else the entire solution to the problem may be “hidden” in these phases). For $k = 1, \dots, c$, let $\tau_P^{(k)}$ be the transformation that produces $P^{(k)}$ from P , and let $\tau_T^{(k)}$ be the transformation that produces $T^{(k)}$ from T . We assume that $\tau_P^{(k)}$ gets as input only the array P , and that it can perform any computation on P . Formally, $\tau_P^{(k)}$ is a function $\tau_P^{(k)} : \Sigma_P^* \rightarrow N^*$. For the text preprocessing transformations we further allow the transformations $\tau_T^{(k)}$ to be dependent on the pattern P . Thus, $\tau_T^{(k)} : \Sigma_T^* \times \Sigma_P^* \rightarrow N^*$. However, we require that the transformation be *local*, in the sense that for any location i , the value $T^{(k)}[i]$ at that location is dependent only on the value of $T[i]$ and not on other locations of T . Formally, the function $\tau_T^{(k)}$ is *local* if for each i there exists a function f_i such that for any T and P , $\tau_T^{(k)}(T, P)[i] = f_i(T[i], P)$. Clearly, we also require that $\tau_T^{(k)}$ be *length preserving*, in the sense that for any T, P , the number of elements in the array $\tau_T^{(k)}(T, P)$ is equal to that in T .

For the postprocessing phase, we allow any computation on values $C^{(1)}[i], \dots, C^{(c)}[i]$ for each i .

We are now ready to formally define the convolutions model.

DEFINITION 2. An algorithm in the convolution model is a quadruplet $A = (c, \tau_P, \tau_T, D)$ where

1. $c = c(m, n)$ is an integer function (where $m = |P|$ and $n = |T|$, the sizes of P and T , respectively),
2. $\tau_P = (\tau_P^{(1)}, \tau_P^{(2)}, \dots)$ is the sequence of pattern preprocessing functions, $\tau_P^{(k)} : \Sigma_P^* \rightarrow N^*$,
3. $\tau_T = (\tau_T^{(1)}, \tau_T^{(2)}, \dots)$ is the sequence of local, length preserving text preprocessing functions, $\tau_T^{(k)} : \Sigma_T^* \times \Sigma_P^* \rightarrow N^*$.
4. $D = D_0, D_2, \dots$ is the sequence of decision criteria, $D_i : N^* \rightarrow \{0, 1\}$.

Given an algorithm A in the convolution model, text T , and pattern P , with $|T| = n$ and $|P| = m$, a computation in the convolutions model is composed on the following three phases:

1. Preprocessing: for each $k = 1, \dots, c$, compute $T^{(k)} = \tau_T^{(k)}$ and $P^{(k)} = \tau_P^{(k)}$.
2. Convolution: for each $k = 1, \dots, c$, compute $C^{(k)} = T^{(k)} \otimes P^{(k)}$.
3. Decide: for each $i = 0, \dots, n - 1$, compute $d_i = D_i(C^{(1)}[i], \dots, C^{(c)}[i])$.

Example. Consider the problem of exact string matching with “don’t cares.” The input to the problem is a pattern array $P \in \{0, 1, \phi\}^m$ and a text array $T \in \{0, 1\}^n$. There is a match at location i if, for $j = 0, \dots, m - 1$, either $P[j] = T[i + j]$ or $P[j] = \phi$. Fischer and Paterson [19] provide the following convolution-based solution to this matching problem. First, compute the following two convolutions:

$$\begin{aligned} C^{(1)} &\leftarrow \chi_0(T) \otimes \chi_1(P), \\ C^{(2)} &\leftarrow \chi_1(T) \otimes \chi_0(P) \end{aligned}$$

(where for $b = 0, 1$, $\chi_b(A)$ is the characteristic function of b). The text locations i , where $C^{(1)}[i] = C^{(2)}[i] = 0$, are those where there is a match with “don’t cares.”

In order to cast this algorithm into the structure defined in Definition 2 set

- $c(m, n)$ is the constant function 2.
- $\tau_P^{(1)} = \chi_1$ and $\tau_P^{(2)} = \chi_0$.
- $\tau_T^{(1)}(T, P) = \chi_0(T)$ and $\tau_T^{(2)}(T, P) = \chi_1(T)$, for all T and P . Note that these functions are indeed local and length preserving.
- for each i ,

$$D_i(C^{(1)}[i], C^{(2)}[i]) = \begin{cases} 1, & C^{(1)}[i] = C^{(2)}[i] = 0, \\ 0, & \text{otherwise.} \end{cases}$$

Our solutions in sections 3 and 4 are all algorithms in the convolutions model.

6.2. The lower bound. We wish to bound the amount of work necessary for solving the function matching problem in the convolution model. We do so by providing a bound on the total number of bits produced in the convolution phase. For a variable x , let $sizeof(x)$ be the number of bits used to represent x (e.g., a bit, a byte, a word).

CLAIM 1. *Let $A = (c, \tau_P, \tau_T, D)$ be an algorithm that solves the function matching problem in the convolutions model. Then, for any m (even) and any n , there exists a pattern P and text T , with $|P| = m$ and $|T| = n$, such that the following holds. For text location i set $s(i) = \sum_{k=1}^c sizeof(C^{(k)}[i])$ ($s(i)$ is the total number of bits in the convolution results for location i). Then, for any $i < n - m$, $s(i) \geq m/2$.*

Proof. Consider the algorithm A from Claim 1 above for computing the function matching in the convolution model. We show how to use A in order to solve the *word equality problem* in the communication complexity setting. We then use known bounds from communication complexity to bound $s(i)$. Suppose that Alice has string $w_1 \in \{0, 1\}^\ell$ and Bob has string $w_2 \in \{0, 1\}^\ell$, and that they wish to determine whether $w_1 = w_2$. Let $w = w_1 w_2$ (the concatenation of the two) and consider the pattern $P = (1, 2, \dots, \ell, 1, 2, \dots, \ell)$. Then, P function-matches with w iff $w_1 = w_2$. Consider a location i . For any k ,

$$C^{(k)}[i] = \left(\sum_{j=i}^{i+\ell-1} T^{(k)}[j] \cdot P^{(k)}[j] \right) + \left(\sum_{j=i+\ell}^{i+2\ell-1} T^{(k)}[j] \cdot P^{(k)}[j] \right).$$

Denote $C_A^{(k)}[i] = \sum_{j=i}^{i+\ell-1} T^{(k)}[j] \cdot P^{(k)}[j]$ (the first summand) and $C_B^{(k)} = \sum_{j=i+\ell}^{i+2\ell-1} T^{(k)}[j] \cdot P^{(k)}[j]$ (the second summand). Since the transformation $T^{(k)}$ is *local*, the value of $C_A^{(k)}[i]$ depends only on the values $T[j]$ with $i \leq j < i + \ell$ and possibly on P , while $C_B^{(k)}[i]$ depends only on the values $T[j]$ with $i + \ell \leq j < i + 2\ell$ and possibly on P . Thus, setting $T_i^{i+\ell-1} = w_1$, $T_{i+\ell}^{i+2\ell-1} = w_2$ and fixing $P = (1, 2, \dots, \ell, 1, 2, \dots, \ell)$, we get that Alice can compute *on her own* $C_A^{(k)}[i]$ (for all k), and similarly Bob can compute *on his own* $C_B^{(k)}[i]$ (for all k). Now, in order to determine if $w_1 = w_2$, we must determine if there is a function match at location i . To this end, Bob sends Alice $C_B^{(k)}[i]$ for all k . Alice then computes $C_A^{(k)}[i]$ and $C^{(k)}[i] = C_A^{(k)}[i] + C_B^{(k)}[i]$, for

$k = 1, \dots, c$, and finally $d = D_i(C^{(1)}[i], \dots, C^{(c)}[i])$ to obtain the result. Thus, the total communication is

$$\sum_{j=1}^c \text{sizeof}(C_B^{(k)}[i]) \leq \sum_{k=1}^c \text{sizeof}(C^{(k)}[i]) = s(i).$$

However, it is known that any protocol for solving the word equality problem for ℓ -bit words in the communication complexity model requires at least ℓ bits [32]. Thus, $s(i) \geq \ell = m/2$. \square

We obtain the following.

THEOREM 2. *Any algorithm for solving the function matching problem in the convolution model requires $\Omega(mn)$ bit operations.*

Proof. By Claim 1, for each $i < n - m$ at least $m/2$ bits must be produced. Thus, in total at least $(n - m)m/2 = \Omega(mn)$ bits must be produced. Producing a bit requires at least one bit operation. \square

The reader is cautioned that the above lower bound does not provide a deterministic lower bound for the function matching problem in the RAM model, but rather only in the convolutions model. While convolutions seem to be the only known effective tool in pattern matching for solving generalized problems such as the function matching problem, the convolutions model is very restrictive. The lower bound does indicate, however, that using convolutions in the way defined in the convolutions model is a direction one should *not follow* when seeking an efficient algorithm for the function matching problem.

7. Two-dimensional parameterized matching. The one-dimensional parameterized matching problem was efficiently solved in [7]. However, as discussed in [4], the move to two dimensions implies a possible computational difficulty if no separable attributes exist. Parameterized matching is not separable—if all columns (or rows) parameterize-match, it does not necessarily imply that the entire matrix parameterize-matches. Thus we are forced to seek other approaches.

Our problem.

INPUT: Two-dimensional pattern P of size $m \times m$ and two-dimensional text T of size $2m \times 2m$.

OUTPUT: All locations $[i, j]$ in T where there is a parameterized occurrence of the pattern.

We may assume the text size $n^2 \leq 4m^2$ because of an argument similar to the one in Observation 2.

Solution idea. We will do a function matching of the pattern in the text. Following that, because of Observation 1 we discard all locations where the number of symbols in the text submatrix is different from $|\Sigma_P|$.

The solution's implementation uses a well-established two-dimensional linearization technique (see, e.g., [8]). Construct a text string T' of length n^2 and a pattern string P' of length m^2 such that a P' occurrence in T' will indeed mean a P occurrence in T .

String T' is taken as the concatenation of all rows of T . P' needs to be constructed more carefully since an occurrence of P in T means that each row of P occurs in T at a distance $n - m$ from its previous row. We can make this happen by introducing a new symbol, ϕ , that does not appear in the alphabets. This symbol's semantics is different from the other alphabet symbols in that it *matches every symbol*. This symbol is called the *don't care* or *wildcard* symbol. Fischer and Paterson [19] used

convolutions to solve the *pattern matching with “don’t cares”* problem. Recall that our algorithms for function matching are convolutions-based.

We have reduced the two-dimensional parameterized matching problem to the *string function matching problem with “don’t cares” in the pattern*.

DEFINITION. *The string function matching problem with “don’t cares” in the pattern is the following:*

INPUT: *Pattern string $P = P[0], \dots, P[m - 1]$ over alphabet $\Sigma_P \cup \{\phi\}$ and text string $T = T[0], \dots, T[n - 1]$ over alphabet Σ_T .*

OUTPUT: *All locations i in T where there is a function occurrence of P , where the symbol ϕ matches everything.*

7.1. Randomized two-dimensional parameterized matching. The only change necessary for Algorithm B to solve the function matching with “don’t cares” in the pattern problem is in the f_P function. Currently every symbol that appears more than once was mapped to a random number between 1 and n^2 . Symbols that appear in the pattern once were always mapped to 0 so they have no effect on the results. The “don’t care” symbol behaves the same way—it is allowed to match every symbol. Therefore for every mapping f_P that we randomly construct, we always have $f_P(\phi) = 0$.

Time. $O(n \log n)$.

COROLLARY 3. *Two-dimensional function matching can be done in time $O(n^2 \log n)$ and with probability $\frac{1}{n^2}$ of false positive matches.*

We still need to count the number of different symbols in the $m \times m$ submatrix at every location where there is a function matching. The problem we need to solve is the following.

SUBMATRIX CHARACTER COUNT problem.

INPUT: *Two-dimensional $n \times n$ array T of symbols over alphabet Σ and a natural number m .*

OUTPUT: *For every $i, j \in \{1, \dots, n - m + 1\}$, the number of different alphabet symbols occurring in the submatrix $T[k, \ell]$, $k = i, \dots, i + m - 1$, $\ell = j, \dots, j + m - 1$.*

Amir, Church, and Dar [4] show that the submatrix character count problem can be solved deterministically in time $O(n^2 \log m)$. We conclude with the following.

COROLLARY 4. *Two-dimensional function matching can be done in time $O(n^2 \log n)$ and with probability $\frac{1}{n^2}$ of false positive matches.*

7.2. Deterministic two-dimensional parameterized matching. It is sufficiently efficient to solve the string function matching problem with “don’t cares” in the pattern. Unfortunately, we do not know how to compute string function matching for general alphabets in $o(nm)$ time. However, our real aim is two-dimensional *parameterized* matching. Therefore our algorithm will make use of both these facts. The result of our algorithm is a set of locations that is a superset of the parameterized match and a subset of the function match. We narrow it down to the parameterized match by counting the number of different symbols.

Recall, then, that we have a linearized T and P , where the length of T is $4m^2$ and the length of P is $2m^2$ (caused by the addition of the “don’t cares” padding).

The algorithm’s idea is to treat symbols that appear many times in the text and pattern (and thus are few in number) differently from the possibly numerous alphabet symbols that appear rarely. The thinking is that convolutions can be used on a small number of symbols (the frequent ones) while some other technique will be effective on the rare symbols, even though they are many.

DEFINITION. *An alphabet symbol $a \in \Sigma_T$ or $a \in \Sigma_P$ is frequent if it appears in*

T or P , respectively, at least m times. Otherwise the symbol is rare.

Our algorithm will treat separately the following three types of symbols: the frequent in the pattern symbols versus the text; the frequent in the text versus the pattern; and then the rare symbols in both text and pattern. For ease of description, we will henceforth say that symbol a function-matches the text at location i , by which we mean that starting at location i for the length of $2m^2$ elements, all text symbols corresponding to an occurrence of a in the pattern are equal. In other words, if there is no function occurrence at location i , then it is not because of the symbol a . We are now ready to provide the outline of the algorithm for two dimensional parameterized matching.

ALGORITHM C

1. Find all text locations where the symbols that are frequent in the pattern function-match.
2. Find all text locations where the symbols that are frequent in the text parameterize-match.
3. Find all text locations where symbols that are rare in the pattern function-match the symbols that are rare in the text.
4. Retain only the text locations where the number of different symbols is precisely equal to the number of different symbols in the pattern.

END

The submatrix character count (step C.4) can be done in time $O(n^2 \log m)$ [4]. The next three subsections show efficient implementations to steps C.1.–C.3.

7.2.1. Frequent pattern symbols. For every frequent symbol a in the pattern, Lemma 2 allows us to find all text location where a function-matches in the manner described in section 4 using a constant number of convolutions. Mark all text location where there was no function-matching, i.e., where a corresponds to at least two distinct text alphabet symbols.

At the end of this stage, replace all frequent pattern symbols with ϕ .

Time. Since there are $O(m)$ frequent pattern symbols, the time for this stage is $O(n^2 m \log m)$.

7.2.2. Frequent text symbols. We desire to find the locations i of the linearized text (corresponding to text locations (l, j)) where the pattern parameterize-matches on the frequent symbols. To do so, remember that the padded pattern is of length $2m^2$ and that we need to find out whether the frequent text symbols in the $2m^2$ substring, of the linearized text, starting at i parameterize-match the pattern. In other words, no frequent text symbol corresponds to more than one pattern symbol. Again, we would like to use Lemma 2. However, now the roles of T and P are reversed. Additional complications are that the number of elements in the sum is different for every location (this number is dependent on the distribution of elements in the text substring) and that there are “don’t cares” in the pattern. Both these problems are solved via a single convolution.

Assume that $\Sigma_P = \{1, \dots, |\Sigma_P|\}$. If it does not, this can always be obtained in time $O(m^2 \log m)$. Let P_2 be the array of length $2m^2$, where $P_2[i] = (P[i])^2$, $i = 0, \dots, 2m^2 - 1$.

Let $K \leftarrow (\chi_a(T) \otimes \chi_{\bar{\phi}}(P))$, where

$$\chi_{\bar{\phi}}(x) = \begin{cases} 1 & \text{if } x \neq \phi, \\ 0 & \text{if } x = \phi. \end{cases}$$

The number of non- ϕ pattern elements that correspond to frequent text symbol a at location i is $K[i]$.

Let $SA \leftarrow \chi_a(T) \otimes P$. By Lemma 2, the locations i , where $(SA[i])^2 = K[i]P_2[i]$, are precisely the locations where all text a 's are matched to a single pattern alphabet symbol.

Note that it may still be the case that one pattern symbol matches the two frequent text symbols a_1 and a_2 . We need to discard these locations as well. This is done as follows. For every location i where all occurrences of a correspond to a single parameter alphabet symbol, find that symbol. This can be found by two convolutions.

Let $N_a \leftarrow \chi_a(T) \otimes \chi_{\bar{\phi}}(P)$. $N_a[i]$ is precisely the number of a 's in the substring of length sm^2 starting at location i that do not correspond to ϕ in the pattern. Assume that $\Sigma_P = \{1, \dots, |\Sigma_P|\}$. If it does not, this can always be obtained in time $O(m \log m)$. Let $C_a \leftarrow \chi_a(T) \otimes P$. For every undiscarded location i , $C_a[i]/N_a[i]$ is the pattern symbol that corresponds to all occurrences of a in the $O(sm^2)$ substring beginning at i .

For every location i where a parameterize-matches with a corresponding pattern symbol b_a , insert b_a in a data structure D_i that holds all different symbols that parameterize-match a frequent text symbol in the substring starting at i . If b_a already appears in the data structure (i.e., it parameterize-matched some other symbol), then mark i as a nonmatch location.

D_i 's size is at most $O(m)$ since there are no more than $O(m)$ frequent symbols. We use a data structure, e.g., balanced trees, that supports *insert* and *search* in time $O(\log m)$. Therefore this stages adds a cost of $O(m \log m)$ for every text location, for a total cost of $O(n^2 m \log m)$.

After all frequent text symbols are handled, mark them all as ϕ . At the end of this stage, all locations where there was no parameterized match of the frequent text symbols are marked.

Time. The time for this stage is dominated by the function matching over the binary alphabets, making the total time $O(n^2 m \log^2 m)$.

7.2.3. Symbols rare in both text and pattern. At this point the only cases we have not handled are correspondences of rare pattern symbols and rare text symbols. There may be many such elements, but each of them appears less than m times. We use *subset matching*, defined by Cole and Hariharan [15], to solve this case.

We construct a new pattern P' and a new text T' in which every element P' and T' is a subset of Z , where Z is the set of integers. In this case, the lengths of P' and T' are the same as the lengths of P and T , respectively.

$P'[i] = \phi$ (the empty set) if $P[i] = \phi$ (the “don't care” symbol).

$T'[i] = Z$ (the universal set) if $T[i] = \phi$ (the “don't care” symbol).

Assume $P[i] = a$ (a rare symbol). Further, assume that a occurs at locations $i_0, i_1, \dots, i_j = i, \dots, i_k$. Then the set $P'[i]$ is $\{i_0 - i, i_1 - i, \dots, i_{j-1} - i, i_{j+1} - i, \dots, i_k - i\}$. In other words, we consider i as the center and write the distance between it and every other occurrence of a in the pattern. The distance to the a occurrences that precede i is negative, and the distance to the a occurrences that follow i is positive.

If $T[i]$ is a rare symbol, then we construct $T'[i]$ in a manner identical to the construction of P' .

P' and T' can be constructed in time $O(n^2 m)$.

Recall that P and T now include only rare symbols. All frequent symbols previously handled and replaced by “don't care” symbols.

DEFINITION. *There is a function occurrence of a rare symbol a of P with the*

rare symbols of T at location i , if one of two conditions hold:

1. Every occurrence of a corresponds to ϕ .
2. Every occurrence of a corresponds to b_a , where b_a is a rare text symbol.

THEOREM 3. *There is a function match of the rare symbols of P in the rare symbols of T iff there is a subset matching of P' in T' .*

Proof. The rare symbols of P function-match the rare symbols of T iff, at every text location i , every occurrence of pattern alphabet symbol a corresponds either to the same text alphabet symbol or to ϕ . If the corresponding text symbol everywhere is ϕ , then it is replaced by Z (the universal set) and thus the subset that replaces a in P' is clearly a subset of the corresponding universal set Z . Otherwise, the alphabet symbol b_a is a rare symbol. a function-matches T at i iff every location where there is an a corresponds to the same b_a . This precisely means that the relative locations of a are a subset of the relative location of b_a , which means that the set in P' that replaces a is a subset of the set in T' that replaces b_a .

Note that the cases where some occurrences of a match a ϕ in the text and the rest match a single b_a in the text are not defined as a function match of rare symbols in rare symbols. The definition follows our problem's requirement since a matching both ϕ and b_a means that some occurrences of a match a frequent text symbol and some match a nonfrequent symbol, and thus it is not a function matching). However, in this case there is not a subset matching either because the text subset of the rare symbols will *not* include the indices of the occurrences corresponding to the Z sets, and thus they will not match. \square

The only thing left for us to do now is analyze the time it takes for subset matching with “don't cares.”

Cole, Hariharan, and Indyk [16] showed that subset matching, in which the pattern has m sets, the text has n sets, and the size of the sets is bound globally by z , can be achieved deterministically in time $O(nz \log^2 m \log \log m)$. This was done by creating a code of size $z \log m \log \log m$. An empty set in the pattern is always easily handled, since it means there are no elements at all. However, in the case of bounded size sets, a universal set is also easily implementable, since it means we have a set of all 1's in those text locations during all convolutions, forcing always a match.

We conclude with the following.

COROLLARY 5. *The time for computing the function-matching of the rare elements is $O(n^2 m \log^2 m \log \log m)$.*

REFERENCES

- [1] K. ABRAHAMSON, *Generalized string matching*, SIAM J. Comput., 16 (1987), pp. 1039–1051.
- [2] A. AMIR, Y. AUMANN, G. LANDAU, M. LEWENSTEIN, AND N. LEWENSTEIN, *Pattern matching with swaps*, J. Algorithms, 37 (2000), pp. 247–266.
- [3] A. AMIR, G. BENSON, AND M. FARACH, *An alphabet independent approach to two-dimensional pattern matching*, SIAM J. Comput., 23 (1994), pp. 313–323.
- [4] A. AMIR, K. W. CHURCH, AND E. DAR, *The submatrices character count problem: An efficient solution using separable values*, Inform. Comput., 190 (2002), pp. 100–116.
- [5] A. AMIR, R. COLE, R. HARIHARAN, M. LEWENSTEIN, AND E. PORAT, *Overlap matching*, in Proceedings of the 12th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA), ACM, New York, SIAM, Philadelphia, 2001, pp. 279–288.
- [6] A. AMIR AND M. FARACH, *Efficient 2-dimensional approximate matching of half-rectangular figures*, Inform. Comput., 118 (1995), pp. 1–11.
- [7] A. AMIR, M. FARACH, AND S. MUTHUKRISHNAN, *Alphabet dependence in parameterized matching*, Inform. Process. Lett., 49 (1994), pp. 111–115.

- [8] A. AMIR AND G. LANDAU, *Fast parallel and serial multidimensional approximate array matching*, Theoret. Comput. Sci., 81 (1991), pp. 97–115.
- [9] A. APOSTOLICO AND Z. GALIL, EDs., *Pattern Matching Algorithms*, Oxford University Press, Oxford, UK, 1997.
- [10] G. P. BABU, B. M. MEHTRE, AND M. S. KANKANHALLI, *Color indexing for efficient image retrieval*, Multimedia Tools Appl., 1 (1995), pp. 327–348.
- [11] B. S. BAKER, *A theory of parameterized pattern matching: Algorithms and applications*, in Proceedings of the 25th Annual ACM Symposium on Theory of Computing, ACM, New York, 1993, pp. 71–80.
- [12] J. H. BOWIE, R. LUTHY, AND D. EISENBERG, *A method to identify protein sequences that fold into a known three-dimensional structure*, Science, 253 (1991), pp. 164–176.
- [13] R. S. BOYER AND J. S. MOORE, *A fast string searching algorithm*, Comm. ACM, 20 (1977), pp. 762–772.
- [14] M. T. CHEN AND J. SEIFERAS, *Efficient and elegant subword-tree construction*, in Combinatorial Algorithms on Words, A. Apostolico and Z. Galil, eds., NATO, Adv. Sci. Inst. Ser. F Comput. Systems Sci. 12, Springer, Berlin, 1985, pp. 97–107.
- [15] R. COLE AND R. HARIHARAN, *Tree pattern matching and subset matching in randomized $O(n \log^3 m)$ time*, in Proceedings of the 29th Annual ACM Symposium on Theory of Computing, ACM, New York, 1997, pp. 66–75.
- [16] R. COLE, R. HARIHARAN, AND P. INDYK, *Tree pattern matching and subset matching in deterministic $O(n \log^3 n)$ -time*, in Proceedings of the 10th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA), ACM, New York, SIAM, Philadelphia, 1999, pp. 245–254.
- [17] T. H. CORMEN, C. E. LEISERSON, AND R. L. RIVEST, *Introduction to Algorithms*, MIT Press, Cambridge, MA; McGraw-Hill, New York, 1990.
- [18] M. CROCHEMORE AND W. RYTTER, *Text Algorithms*, Oxford University Press, Oxford, UK, 1994.
- [19] M. J. FISCHER AND M. S. PATERSON, *String matching and other products*, in Complexity of Computation, R. M. Karp, ed., SIAM-AMS Proceedings 7, AMS, Providence, RI, 1974, pp. 113–125.
- [20] D. E. KNUTH, J. H. MORRIS, JR., AND V. R. PRATT, *Fast pattern matching in strings*, SIAM J. Comput., 6 (1977), pp. 323–350.
- [21] S. RAO KOSARAJU, *Efficient String Matching*, manuscript, 1987.
- [22] W. C. KREHLING AND C. NORRIS, *Profile assisted register allocation*, in Proceedings of the ACM Symposium on Applied Computing (SAC), ACM, New York, 2000, pp. 774–781.
- [23] G-Y. LUEH, T. GROSS, AND A-R. ADL-TABATABAI, *Fusion-based register allocation*, ACM Trans. Programming Languages and Systems, 22 (2000), pp. 431–470.
- [24] K. MERZ, JR., AND S. M. LE GRAND, *The Protein Folding Problem and Tertiary Structure Prediction*, Birkhäuser Boston, Boston, MA, 1994.
- [25] S. MUTHUKRISHNAN, *New results and open problems related to nonstandard stringology*, in Proceedings of the 6th Annual Combinatorial Pattern Matching Conference, Lecture Notes in Comput. Sci. 937, Springer, Berlin, 1995, pp. 298–317.
- [26] S. MUTHUKRISHNAN AND K. PALEM, *Nonstandard stringology: Algorithms and complexity*, in Proceedings of the 26th Annual Symposium on the Theory of Computing, ACM, New York, 1994, pp. 770–779.
- [27] M. SWAIN AND D. BALLARD, *Color indexing*, Internat. J. Comput. Vision, 7 (1991), pp. 11–32.
- [28] U. VISHKIN, *Optimal parallel pattern matching in strings*, in Inform. and Control, 67 (1985), pp. 91–113.
- [29] U. VISHKIN, *Deterministic sampling—a new technique for fast pattern matching*, SIAM J. Comput., 20 (1991), pp. 22–40.
- [30] P. WEINER, *Linear pattern matching algorithm*, in Proceedings of the 14th Annual IEEE Symposium on Switching and Automata Theory, IEEE, Piscataway, NJ, 1973, pp. 1–11.
- [31] J. YADGARI, A. AMIR, AND R. UNGER, *Genetic algorithms for protein threading*, in Proceedings of the 6th Annual International Conference on Intelligent Systems for Molecular Biology (ISMB 98), J. Glasgow, T. Littlejohn, F. Major, R. Lathrop, D. Sankoff, and C. Sensen, eds., AAAI Press, Menlo Park, CA, 1998, pp. 193–202.
- [32] A. C. C. YAO, *Some complexity questions related to distributive computing*, in Proceedings of the Eleventh Annual Symposium on the Theory of Computing (STOC), ACM, New York, 1979, pp. 209–213.

FINDING FOUR INDEPENDENT TREES*

SEAN CURRAN[†], ORLANDO LEE[‡], AND XINGXING YU[§]

Abstract. Motivated by a multitree approach to the design of reliable communication protocols, Itai and Rodeh gave a linear time algorithm for finding two independent spanning trees in a 2-connected graph. Cheriyan and Maheshwari gave an $O(|V|^2)$ algorithm for finding three independent spanning trees in a 3-connected graph. In this paper we present an $O(|V|^3)$ algorithm for finding four independent spanning trees in a 4-connected graph. We make use of chain decompositions of 4-connected graphs.

Key words. Connectivity, chain decomposition, numbering, independent trees, algorithm

AMS subject classifications. 05C40, 05C85, 05C38, 05C75

DOI. 10.1137/S0097539703436734

1. Introduction. We consider simple graphs only. For a graph G , we use $V(G)$ and $E(G)$ to denote the vertex set and edge set of G , respectively.

For a tree T and $x, y \in V(T)$, let $T[x, y]$ denote the unique path from x to y in T . A *rooted tree* is a tree with a specified vertex called the *root* of T . Let G be a graph, let $r \in V(G)$, and let T and T' be trees of G rooted at r . We say that T and T' are *independent* if for every $x \in V(T) \cap V(T')$, the paths $T[r, x], T'[r, x]$ have no vertex in common except r and x .

The study of independent spanning trees started with Itai and Rodeh [11], where they proposed a multitree approach to reliability in distributed networks (see also [5]). They developed a linear time algorithm that, given any vertex r in a 2-connected graph G , finds two independent spanning trees of G rooted at r . Later, Cheriyan and Maheshwari [1] proved that for any vertex r in a 3-connected graph G , there exist three independent spanning trees of G rooted at r . Furthermore, they gave an $O(|V(G)|^2)$ algorithm for finding these trees.

Itai and Zehavi [12] also proved that every 3-connected graph contains three independent spanning trees (rooted at any vertex), and they conjectured that for any k -connected graph G and for any $r \in V(G)$, there exist k independent spanning trees of G rooted at r . According to Schrijver [14], the Itai-Zehavi conjecture is part of a more general conjecture by Frank [6]. Huck [9] proved this conjecture for planar 4-connected graphs. Later, Miura et al. [13] gave a linear time algorithm for finding four independent rooted spanning trees in a planar 4-connected graph.

Our main result is the following.

*Received by the editors October 28, 2003; accepted for publication (in revised form) August 15, 2005; published electronically March 3, 2006.

<http://www.siam.org/journals/sicomp/35-5/43673.html>

[†]School of Mathematics, Georgia Institute of Technology, Atlanta, GA 30332 (curran@math.gatech.edu). The work of this author was partially supported by an NSF VIGRE grant.

[‡]Instituto de Computação, Universidade Estadual de Campinas, Avenida Albert Einstein, 1251, Caixa Postal 6176, 13083-971 Campinas - SP, Brazil (lee@ic.unicamp.br). The work of this author was supported by CNPq (Proc: 200611/00-3)—Brazil.

[§]School of Mathematics, Georgia Institute of Technology, Atlanta, GA 30332 and Center of Combinatorics, LPMC, Nankai University, Tianjin 300071, China (yu@math.gatech.edu) The work of the third author was partially supported by NSF grant DMS-0245530 and by NSA grant MDA-904-03-1-0052.

THEOREM 1.1. *Let G be a 4-connected graph, and let $r \in V(G)$. Then there exist four independent spanning trees of G rooted at r . Moreover, such trees can be found in $O(|V(G)|^3)$ time.*

To provide motivation for our method, we first describe Itai and Rodeh's method for constructing two independent spanning trees rooted at a vertex r in a 2-connected graph. Let G be a 2-connected graph, and let r and t be two adjacent vertices of G . An r - t numbering is a function $g : V(G) \mapsto \{1, \dots, n\}$ with $n \geq |V(G)|$ satisfying the following properties:

- (i) $g(r) = 1$ and $g(t) = n$.
- (ii) Every vertex $v \in V(G) - \{r, t\}$ has a neighbor u with $g(u) < g(v)$ and a neighbor w with $g(w) > g(v)$.

An r - t numbering can be produced from an ear decomposition of G . From an r - t numbering g , Itai and Rodeh define two independent spanning trees T_1 and T_2 of G rooted at r as follows. For each vertex $v \in V(G) - \{r\}$, specify its parent in each tree. In tree T_1 , for each $v \in V(G) - \{r\}$, the parent of v is a neighbor u for which $g(u) < g(v)$. In tree T_2 , the parent of t is r and, for each $v \in V(G) - \{r, t\}$, the parent of v is a neighbor w for which $g(w) > g(v)$. It is not hard to show that T_1 and T_2 are independent spanning trees in G rooted at r .

The idea for constructing four independent spanning trees in a 4-connected graph is inspired by the 2-connected case. The main difference is that we need to use two numberings instead of one. This idea can be roughly described as follows. Let G be a 4-connected graph, and let $r \in V(G)$. First, we compute a decomposition of G into "planar chains," a generalization of ear decomposition, which we describe in section 2. From this decomposition, we find two numberings g and f . We then construct these trees from these numberings.

The main difficulty with this idea lies in the fact that it is not possible to number all vertices of G , because the "chains" in our decomposition need not be paths. Fortunately, the nonpath part of the chains are planar, and we can compute four independent spanning trees in each one of these planar parts using the algorithm of Miura et al. [13] mentioned above. These trees are then used to number every vertex in the planar parts that has neighbors outside its chain. Once these numberings are computed, we can construct four independent spanning trees.

The rest of this paper is organized as follows. The remainder of this section is devoted to notation and terminology. In section 2 we describe chain decomposition of a graph and state the main decomposition result from [4] (also see [3]). In section 3 we describe known results for the planar case and give some auxiliary lemmas. In section 4 we give algorithms for constructing the required numberings. In section 5 we describe an algorithm for constructing four independent spanning trees in a 4-connected graph, and we prove its correctness in section 6.

Throughout this paper, we use $A := B$ to rename B as A or to define A as B . We use the notation xy (or yx) to represent an edge with ends x and y . Let G be a graph. For any $S \subseteq V(G)$, let $G[S]$ denote the subgraph of G with $V(G[S]) = S$ and $E(G[S])$ consisting of the edges of G with both ends in S ; we say that $G[S]$ is the subgraph of G induced by S . Let $G - S := G[V(G) - S]$. A subgraph H of G is an induced subgraph of G if $G[V(H)] = H$. We also say that H is induced in G . For any $H \subseteq G$ and $S \subseteq V(G) \cup E(G)$, $H + S$ denotes the graph with vertex set $V(H) \cup (S \cap V(G))$ and edge set $E(H) \cup \{uv \in S : \{u, v\} \subseteq V(H) \cup (S \cap V(G))\}$.

A graph G is k -connected, where k is a positive integer, if $|V(G)| \geq k + 1$ and, for any $S \subseteq V(G)$ with $|S| \leq k - 1$, $G - S$ is connected. A subgraph H of G is nonseparating in G if $G - V(H)$ is connected.

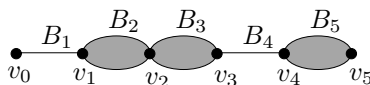


FIG. 1. Example of a chain.

Let G be a graph. For $S \subseteq V(G)$, let $N_G(S) := \{x \in V(G) - S : xy \in E(G) \text{ for some } y \in S\}$. For a subgraph H of G , let $N_G(H) := N_G(V(H))$. When $S = \{x\}$, we let $N_G(x) := N_G(\{x\})$. When there exists no ambiguity, we may simply use $N(S), N(H)$, and $N(x)$, instead of $N_G(S), N_G(H)$, and $N_G(x)$, respectively.

We describe a *path* in G as a sequence $P = (v_1, v_2, \dots, v_k)$ of distinct vertices of G such that $v_i v_{i+1} \in E(G)$, $1 \leq i \leq k - 1$. The vertices v_1 and v_k are called the *ends* of the path P , and the vertices in $V(P) - \{v_1, v_k\}$ are called the *internal vertices* of P . For $1 \leq i \leq j \leq k$, let $P[v_i, v_j] := (v_i, \dots, v_j)$, and for $1 \leq i < j \leq k$, let $P(v_i, v_j) := P[v_{i+1}, v_{j-1}]$. For $A, B \subseteq V(G)$, we say that a path P is an A - B path if one end of P is in A , the other end is in B , and no internal vertex of P is in $A \cup B$. If P is a path with ends a and b , we say that P is a *path from a to b* , or P is an a - b path. Two paths P and Q are *disjoint* if $V(P) \cap V(Q) = \emptyset$. Two paths are *internally disjoint* if no internal vertex of one path is contained in the other. Given a path P in G and a set $S \subseteq V(G)$ (respectively, a subgraph S of G), we say that P is *internally disjoint from S* if no internal vertex of P is contained in S (respectively, $V(S)$). We also describe a *cycle* in G as a sequence $C = (v_1, v_2, \dots, v_k, v_1)$ such that the vertices v_1, \dots, v_k are distinct, $v_i v_{i+1} \in E(G)$, for $1 \leq i \leq k - 1$, and $v_k v_1 \in E(G)$.

2. Chain decomposition. In order to prove Theorem 1.1, we rely on the existence of a *nonseparating chain decomposition* of a 4-connected graph, proved in [4] (also see [3]). Such a decomposition is similar to an ear decomposition. An *ear decomposition* of a graph G is a sequence (P_0, P_1, \dots, P_t) such that (i) P_0 is a cycle in G , (ii) P_1, \dots, P_t are paths in G , (iii) $\bigcup_{i=0}^t P_i = G$, and (iv) for each $0 \leq i \leq t - 1$, $G_i := \bigcup_{j=0}^i P_j$ is 2-connected and $P_{i+1} \cap G_i$ consists of the ends of P_{i+1} . In a nonseparating chain decomposition, the P_i 's will be chains and cycle chains, which may be thought of as a generalization of paths and cycles.

DEFINITION 2.1. A connected graph H is a chain if its blocks can be labeled as B_1, \dots, B_k , where $k \geq 1$ is an integer, and its cut vertices can be labeled as v_1, \dots, v_{k-1} such that

- (i) $V(B_i) \cap V(B_{i+1}) = \{v_i\}$ for $1 \leq i \leq k - 1$, and
- (ii) $V(B_i) \cap V(B_j) = \emptyset$ if $|i - j| \geq 2$ and $1 \leq i, j \leq k$.

We let $H := B_1 v_1 B_2 v_2 \dots v_{k-1} B_k$ denote this situation. If $k \geq 2$, let $v_0 \in V(B_1) - \{v_1\}$ and $v_k \in V(B_k) - \{v_{k-1}\}$, or, if $k = 1$, let $v_0, v_k \in V(B_1)$ with $v_0 \neq v_k$; then we say that H is a v_0 - v_k chain, and we denote this by $H := v_0 B_1 v_1 \dots v_{k-1} B_k v_k$. We usually fix v_0 and v_k , and we refer to them as the ends of H_i . See Figure 1 for an example with $k = 5$.

DEFINITION 2.2. A connected graph H is a cyclic chain if for some integer $k \geq 2$, there exist subgraphs B_1, \dots, B_k of H and vertices v_1, \dots, v_k of H such that

- (i) for $1 \leq i \leq k$, B_i is 2-connected or B_i is induced by an edge of H ,
- (ii) $V(H) = \bigcup_{i=1}^k V(B_i)$ and $E(H) = \bigcup_{i=1}^k E(B_i)$,
- (iii) if $k = 2$, then $V(B_1) \cap V(B_2) = \{v_1, v_2\}$ and $E(B_1) \cap E(B_2) = \emptyset$, and
- (iv) if $k \geq 3$, then $V(B_i) \cap V(B_{i+1}) = \{v_i\}$ for $1 \leq i \leq k$, where $B_{k+1} := B_1$, and $V(B_i) \cap V(B_j) = \emptyset$ for $1 \leq i < i + 2 \leq j \leq k$ and $(i, j) \neq (1, k)$.

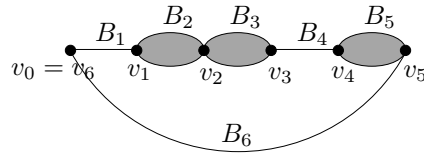


FIG. 2. Example of a cyclic chain.

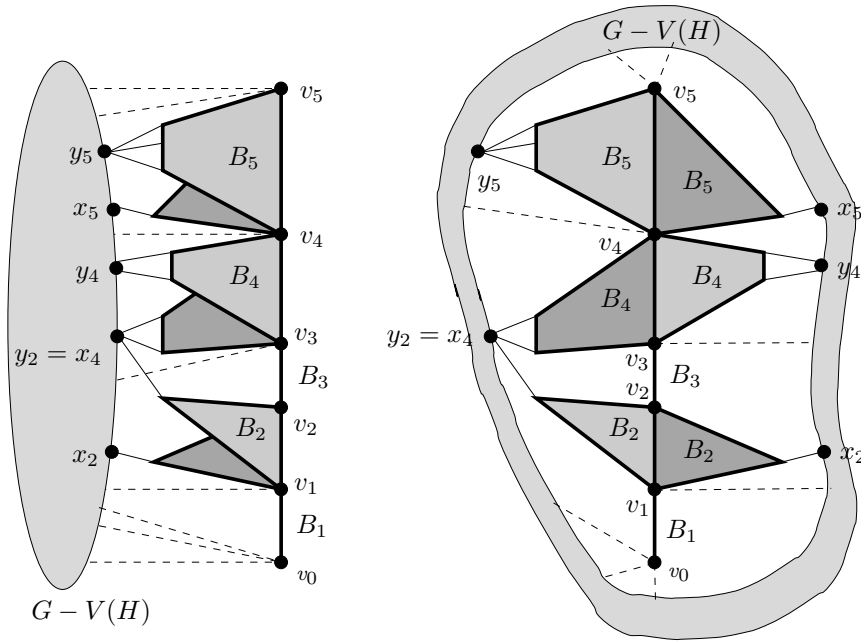


FIG. 3. A planar chain $H := v_0B_1v_1B_2v_2B_3v_3B_4v_4B_5v_5$ in a graph G .

For notational convenience, we usually fix one of the vertices v_1, \dots, v_k as the root of H , say v_k , and write $H := v_0B_1v_1 \dots v_{k-1}B_kv_k$ to indicate that H is a cyclic chain rooted at $v_0 (= v_k)$. Each subgraph B_i is called a piece of H . We sometimes write $I(H) := V(H)$. See Figure 2 for an example with $k = 6$.

In the chain decompositions we will work with, the blocks and pieces have a planar structure. Let G be a graph with distinct vertices a, b, c , and d . We say that the quintuple (G, a, b, c, d) is planar if G can be drawn in a closed disc in the plane with no pair of edges crossing such that a, b, c, d occur in cyclic order on the boundary of the disc. For a graph G and $x, y \in V(G)$, we use $G - xy$ to denote the graph with vertex set $V(G)$ and edge set $E(G) - \{xy\}$ (note that xy need not be an edge of G).

DEFINITION 2.3. Let G be a graph, and let $H := v_0B_1v_1 \dots v_{k-1}B_kv_k$ be a chain (respectively, cyclic chain). If H is an induced subgraph of G , then we say that H is a chain in G (respectively, cyclic chain in G). We say that H is planar in G if, for each $1 \leq i \leq k$ with $|V(B_i)| \geq 3$ (or equivalently, B_i is 2-connected), there exist distinct vertices $x_i, y_i \in V(G) - V(H)$ such that $(G[V(B_i) \cup \{x_i, y_i\}] - x_iy_i, x_i, v_{i-1}, y_i, v_i)$ is planar, and $B_i - \{v_{i-1}, v_i\}$ is a component of $G - \{x_i, y_i, v_{i-1}, v_i\}$. We also say that H is a planar v_0 - v_k chain (respectively, planar cyclic chain). See Figure 3 for two drawings of an example with $k = 5$. The dashed edges may or may not exist, but they are not part of H .

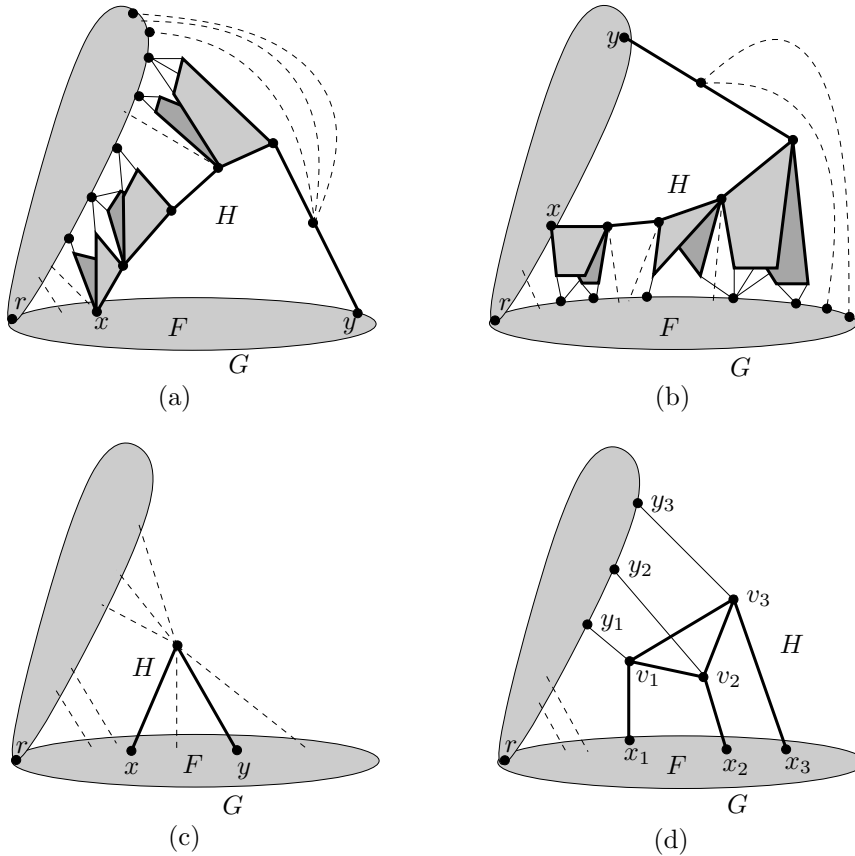


FIG. 4. (a) An up F -chain, (b) a down F -chain, (c) an elementary F -chain, and (d) a triangle F -chain. The dashed edges need not exist.

We can now describe the chains in nonseparating chain decompositions. See Figure 4 for illustrations.

DEFINITION 2.4. Let G be a graph, let F be a subgraph of G , and let $r \in V(F)$. Let H be a planar x - y chain in G such that $V(H) - \{x, y\} \subseteq V(G) - V(F)$. We say that

- (i) H is an up F -chain if $\{x, y\} \subseteq V(F)$ and $N_G(H - \{x, y\}) \subseteq (V(G) - V(F - r)) \cup \{x, y\}$,
- (ii) H is a down F -chain if $\{x, y\} \subseteq V(G) - V(F - r)$ and $N_G(H - \{x, y\}) \subseteq V(F - r) \cup \{x, y\}$, and
- (iii) H is an elementary F -chain if $\{x, y\} \subseteq V(F)$ and H is an x - y path of length two.

In any of the three cases we say that H is a planar x - y F -chain in G (or simply a planar F -chain). Let $I(H) := V(H) - \{x, y\}$.

DEFINITION 2.5. Let G be a graph, let F be a subgraph of G , and let $r \in V(F)$. Suppose that $\{v_1, v_2, v_3\} \subseteq V(G) - V(F)$ induces a triangle T in G and, for each $1 \leq i \leq 3$, v_i has exactly one neighbor x_i in $V(F - r)$ and exactly one neighbor y_i in $V(G) - (V(F) \cup V(T))$, and each v_i has degree four in G . Moreover, assume that x_1, x_2, x_3 are distinct and y_1, y_2, y_3 are distinct. Then we say that $H := T + \{x_1, x_2, x_3, v_1x_1, v_2x_2, v_3x_3\}$ is a triangle F -chain in G . We let $I(H) := \{v_1, v_2, v_3\}$.

Note that Definitions 2.4 and 2.5 depend on the choice of r and F , but in spite of this, whenever we use these concepts in this paper, it should be clear which pair r, F we refer to.

DEFINITION 2.6. *Let G be a graph, let F be a subgraph of G , and let $r \in V(F)$. By a good F -chain in G , we mean an up F -chain or a down F -chain, or an elementary F -chain or a triangle F -chain.*

We are now ready to describe a chain decomposition, which is similar to an ear decomposition.

DEFINITION 2.7. *Let G be a graph, let $r \in V(G)$, and let H_1, \dots, H_t be chains in G , where $t \geq 2$. We say that (H_1, \dots, H_t) is a nonseparating chain decomposition of G rooted at r if the following conditions hold:*

- (i) H_1 is a planar cyclic chain in G rooted at r .
- (ii) For each $i = 2, \dots, t - 1$, H_i is a good $G[\bigcup_{j=1}^{i-1} I(H_j)]$ -chain in G .
- (iii) $H_t := G - (\bigcup_{j=1}^{t-1} I(H_j) - \{r\})$ is a planar cyclic chain in G rooted at r .
- (iv) For each $i = 1, \dots, t - 1$, both $G[\bigcup_{j=1}^i I(H_j)]$ and $G - ((\bigcup_{j=1}^i I(H_j)) - \{r\})$ are 2-connected.

The chains H_2, \dots, H_{t-1} are called internal chains of the nonseparating chain decomposition. If ra is a piece of H_1 , then we say that H_1, \dots, H_t is a nonseparating chain decomposition of G starting at ra .

The following result is proved in [4] (also see [3]).

THEOREM 2.8. *Let G be a 4-connected graph, let $r \in V(G)$, and let $ra \in E(G)$. Then G has a nonseparating chain decomposition rooted at r and starting at ra , and such a decomposition can be found in $O(|V(G)|^2|E(G)|)$ time.*

The basic idea for constructing four independent spanning trees (rooted at r) can be described as follows. By Theorem 2.8, G has a nonseparating chain decomposition (H_1, \dots, H_t) rooted at r . For $1 \leq i \leq t$, let $G_i := G[\bigcup_{j=1}^i I(H_j)]$. We compute two numberings g, f defined on $V(G)$ which resemble r - t numberings. From g we compute two independent spanning trees T_1, T_2 such that for each $i = 1, \dots, t$, the restriction of T_1 and T_2 to G_i are independent spanning trees in G_i rooted at r . Similarly, from f we compute two spanning trees T_3, T_4 such that for each $i = 1, \dots, t$, the restriction of T_3 and T_4 to $G - (V(G_i) - r)$ are independent spanning trees rooted at r .

3. Planar graphs. Let G be a 4-connected graph, and let $r \in V(G)$. To use a nonseparating chain decomposition of G for constructing four independent spanning trees rooted at r , we must be able to find four independent spanning trees in the planar blocks and pieces. Unlike the original problem, these trees are not rooted at the same vertex, but they are rooted at four distinct vertices. Before we describe this result, we introduce some definitions.

DEFINITION 3.1. *Let T and T' be two trees in a graph G with roots r and r' , respectively. We say that T and T' are independent if, for each $x \in V(T) \cap V(T')$, the paths $T[r, x]$ and $T'[r', x]$ have no vertex in common except x (and r if $r = r'$).*

Let G be a graph, and let $S := \{t_1, \dots, t_4\}$ be a set of vertices of G . A 4-tuple $\mathcal{T} := \{T_1, \dots, T_4\}$ is an S -system of G if, for $1 \leq i \leq 4$, T_i is a tree of G rooted at t_i , $V(T_i) \subseteq V(G) - (S - \{t_i\})$, and $t_i \in V(T_i)$. An S -system $\mathcal{T} := \{T_1, \dots, T_4\}$ is independent if the trees in the system are pairwise independent, and an S -system \mathcal{T} is spanning if $V(T_i) = V(G) - (S - \{t_i\})$ for $1 \leq i \leq 4$. See Figure 5 for an example, where the darkened edges are in the trees.

Let G be a graph, let $S \subseteq V(G)$, and let k be a positive integer. We say that G is (k, S) -connected if $|V(G)| \geq |S| + 1$, G is connected, and for any $T \subseteq V(G)$ with $|T| \leq k - 1$, every component of $G - T$ contains an element of S .

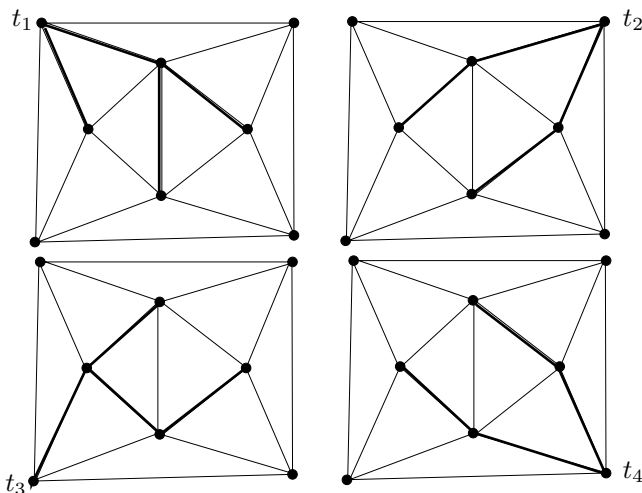


FIG. 5. Four independent trees in a plane graph forming an independent spanning system.

THEOREM 3.2. *Let (G, a, b, c, d) be a planar graph, and suppose that G is $(4, \{a, b, c, d\})$ -connected. Then there exists an independent spanning $\{a, b, c, d\}$ -system of G . Moreover, one can find such a system in linear time.*

The existence of an independent system in Theorem 3.2 was proved by Huck [9]. Huck’s proof is not based on a decomposition of a planar graph, but through a careful analysis of his proof, one can extract an $O(|V(G)|^3)$ algorithm. Miura et al. [13] gave a linear algorithm for finding such a system based on a decomposition of 4-connected planar graphs. In fact, the decomposition they obtained can be viewed as a special case of a nonseparating chain decomposition.

Before we proceed, let us recall that the problem of finding an embedding of a planar graph can be solved in linear time [7, 8]. Moreover, the following problem can be solved in linear time: find a drawing of a planar quintuple (G, a, b, c, d) in a closed disc in the plane with no pair of edges crossing such that a, b, c, d occur in cyclic order on the boundary of the disc. We make no further mention of this fact, but it is implicitly used throughout this section.

In what follows we will use Theorem 3.2 to prove some results concerning “orderings” of certain vertices of a planar graph (G, a, b, c, d) . These results correspond to Lemmas 3.4, 3.5, 3.6, and 3.7. They will be used in the next section to compute two numberings of subsets of $V(G)$.

DEFINITION 3.3. *Let (G, a, b, c, d) be a planar graph, and let $\{T_a, T_b, T_c, T_d\}$ be an independent spanning $\{a, b, c, d\}$ -system of G , where T_v is rooted at v for each $v \in \{a, b, c, d\}$. Let $U \subseteq (N_G(b) \cup N_G(d)) - \{a, c\}$. We say that a permutation u_1, \dots, u_p of the elements of U is a (T_a, T_c) -ordering of U if, for $i, j \in \{1, \dots, p\}$ with $i < j$, $T_a[a, u_i]$ and $T_c[c, u_j]$ are (vertex) disjoint. We also say that u_1, \dots, u_p is (T_a, T_c) -ordered.*

Our first lemma concerns (T_a, T_c) -orderings restricted to elements in $N_G(b) - \{a, c\}$. In this case, this ordering corresponds to a total order.

LEMMA 3.4. *Let (G, a, b, c, d) be a planar graph, and let $\{T_a, T_b, T_c, T_d\}$ be an independent spanning $\{a, b, c, d\}$ -system of G , where T_v is rooted at v for each $v \in \{a, b, c, d\}$. Then there exists a unique (T_a, T_c) -ordering of $N_G(b) - \{a, c\}$. Moreover, such an ordering can be found in $O(|V(G)|)$ time.*

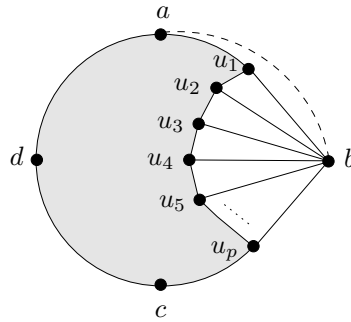


FIG. 6. u_1, \dots, u_p is the unique (T_a, T_c) -ordering of $N_G(b) - \{a, c\}$.

Proof. Let $G' := G - \{ab, bc\}$. If b has at most one neighbor in G' , then the result follows immediately. So assume b has at least two neighbors in G' . Take an embedding of G' in a closed disc such that a, b, c, d occur in clockwise order on the boundary of the disc (such an embedding for G can be computed in linear time). Let u_1, \dots, u_p ($p \geq 2$) denote the neighbors of b in G' in that cyclic order around b such that a, u_1, b, u_p, c, d occur in clockwise order on the boundary of the disc (see Figure 6). Since T_a, T_c are independent, we have that for each $i \in \{1, \dots, p\}$, $T_a[a, u_i]$ and $T_c[c, u_i]$ are internally disjoint. Then by planarity one can see that, for every $i, j \in \{1, \dots, p\}$ with $i \neq j$, $T_a[a, u_i]$ and $T_c[c, u_j]$ are disjoint if and only if $i < j$. Thus, u_1, \dots, u_p is the unique (T_a, T_c) -ordering of $N_G(b) - \{a, c\}$. Clearly, such an ordering can be computed in $O(|V(G)|)$ time. \square

In the next lemma we show that it is possible to extend a (T_a, T_c) -ordering of $N_G(b) - \{a, c\}$ and a (T_a, T_c) -ordering of $N_G(d) - \{a, c\}$ to a (T_a, T_c) -ordering of $(N_G(b) \cup N_G(d)) - \{a, c\}$.

LEMMA 3.5. *Let (G, a, b, c, d) be a planar graph, and let $\{T_a, T_b, T_c, T_d\}$ be an independent spanning $\{a, b, c, d\}$ -system of G , where T_v is rooted at v for each $v \in \{a, b, c, d\}$. Then there exists a (T_a, T_c) -ordering of $(N_G(b) \cup N_G(d)) - \{a, c\}$. Moreover, such an ordering can be found in $O(|V(G)|^2)$ time.*

Proof. Take an embedding of G in a closed disc such that a, b, c, d occur in clockwise order on the boundary of the disc. Consider the following relation. For $u, v \in (N_G(b) \cup N_G(d)) - \{a, c\}$, we say that $u \prec v$ if either one of the following holds:

- (i) $u \in N_G(b)$ and $T_a[a, u] \cap T_c[c, v] = \emptyset$.
- (ii) $u \in N_G(d)$, $T_a[a, u] \cap T_c[c, v] = \emptyset$, and $T_a[a, v] \cap T_c[c, u] \neq \emptyset$.

See Figure 7 for an illustration of conditions (i) and (ii). The bold lines denote the paths in T_a and the dashed lines denote the paths in T_c . Next, we show that \prec defines a total order on $(N_G(b) \cup N_G(d)) - \{a, c\}$.

First, we show that for any distinct $x, y \in (N_G(b) \cup N_G(d)) - \{a, c\}$, either $x \prec y$, or $y \prec x$, but not both. If $x, y \in N_G(b)$ or $x, y \in N_G(d)$, then by planarity, either $T_a[a, x] \cap T_c[c, y] = \emptyset$ and $T_a[a, y] \cap T_c[c, x] \neq \emptyset$, or $T_a[a, x] \cap T_c[c, y] \neq \emptyset$ and $T_a[a, y] \cap T_c[c, x] = \emptyset$. So by (i) or (ii), either $x \prec y$, or $y \prec x$, but not both. Thus, we may assume that $x \in N_G(b)$ and $y \in N_G(d)$. If $T_a[a, x] \cap T_c[c, y] = \emptyset$, then x, y satisfy (i) (as u, v) but not (ii) (as v, u), and we have $x \prec y$ and $y \not\prec x$. So assume $T_a[a, x] \cap T_c[c, y] \neq \emptyset$. Then x, y does not satisfy (i) (as u, v), and hence, $x \not\prec y$. Since T_a and T_c are independent, $T_a[a, y]$ and $T_c[c, y]$ are internally disjoint, and $T_a[a, x]$ and $T_c[c, x]$ are internally disjoint. By planarity, $T_a[a, y] \cap T_c[c, x] = \emptyset$. Therefore, $y \prec x$.

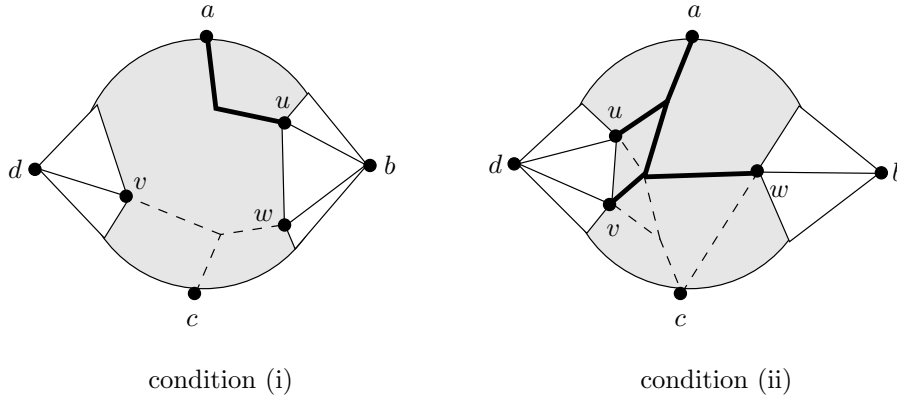


FIG. 7. $u \prec v$ and $u \prec w$.

It remains to show that \prec is transitive. Let $x, y, z \in (N_G(b) \cup N_G(d)) - \{a, c\}$, and assume that $x \prec y$ and $y \prec z$. We will show that $x \prec z$. We have eight cases by considering which of x, y, z are in $N_G(b)$.

- (1) $x, y, z \in N_G(b)$. Since $x \prec y$ and $y \prec z$, it follows from (i) that $T_a[a, x] \cap T_c[c, y] = \emptyset$ and $T_a[a, y] \cap T_c[c, z] = \emptyset$. So by planarity, $T_a[a, x] \cap T_c[c, z] = \emptyset$, and by (i), $x \prec z$.
- (2) $x, y, z \in N_G(d)$. Since $x \prec y$ and $y \prec z$, it follows from (ii) that $T_a[a, x] \cap T_c[c, y] = \emptyset$, $T_a[a, y] \cap T_c[c, x] \neq \emptyset$, $T_a[a, y] \cap T_c[c, z] = \emptyset$, and $T_a[a, z] \cap T_c[c, y] \neq \emptyset$. So by planarity, $T_a[a, x] \cap T_c[c, z] = \emptyset$ and $T_a[a, z] \cap T_c[c, x] \neq \emptyset$. By (ii), $x \prec z$.
- (3) $y, z \in N_G(b)$ and $x \in N_G(d)$. Since T_a and T_c are independent, $P := T_a[a, y] \cup T_c[c, y]$ is an a - c path in $G - \{b, d\}$. Note that P divides the disc into two closed regions, say B and D , with b in B and d in D . Since $x \prec y$ and $x \in N_G(d)$, it follows from (ii) that $T_a[a, x] \cap T_c[c, y] = \emptyset$ and $T_a[a, y] \cap T_c[c, x] \neq \emptyset$. Since $y \prec z$ and $y \in N_G(b)$, it follows from (i) that $T_a[a, y] \cap T_c[c, z] = \emptyset$. So $T_c[c, z]$ lies in B and $T_a[a, x]$ lies in D . Therefore, by planarity, $T_a[a, x] \cap T_c[c, z] = \emptyset$. Since $T_a[a, y] \cap T_c[c, x] \neq \emptyset$, it follows by planarity that $T_a[a, z] \cap T_c[c, x] \neq \emptyset$. Therefore, $x \prec z$.
- (4) $y, z \in N_G(d)$ and $x \in N_G(b)$. Since T_a and T_c are independent, $P := T_a[a, y] \cup T_c[c, y]$ is an a - c path in $G - \{b, d\}$, and P divides the disc into two closed regions B and D , with b in B and d in D . Since $x \prec y$ and $x \in N_G(b)$, it follows from (i) that $T_a[a, x] \cap T_c[c, y] = \emptyset$, and since $y \prec z$ and $y \in N_G(d)$, it follows from (ii) that $T_a[a, y] \cap T_c[c, z] = \emptyset$. So $T_c[c, z]$ lies in D and $T_a[a, x]$ lies in B . By planarity, $T_a[a, x] \cap T_c[c, z] = \emptyset$, and hence by (i), $x \prec z$.
- (5) $x, y \in N_G(d)$ and $z \in N_G(b)$. Since T_a and T_c are independent, $P := T_a[a, y] \cup T_c[c, y]$ is an a - c path in $G - \{b, d\}$, and P divides the disc into two closed regions B and D , with b in B and d in D . Since $x \prec y$ and $x \in N_G(d)$, it follows from (ii) that $T_a[a, x] \cap T_c[c, y] = \emptyset$, and since $y \prec z$ and $y \in N_G(d)$, it follows from (ii) that $T_a[a, y] \cap T_c[c, z] = \emptyset$. Thus, $T_a[a, x]$ lies in D and $T_c[c, z]$ lies in B . By planarity, $T_a[a, x] \cap T_c[c, z] = \emptyset$. Moreover, since $y \prec z$ and $y \in N_G(d)$, it follows from (ii) that $T_a[a, z] \cap T_c[c, y] \neq \emptyset$. By planarity, $T_a[a, z] \cap T_c[c, x] \neq \emptyset$. Therefore by (ii), $x \prec z$.
- (6) $x, y \in N_G(b)$ and $z \in N_G(d)$. Since T_a and T_c are independent, $P := T_a[a, y] \cup T_c[c, y]$ is an a - c path in $G - \{b, d\}$, and P divides the disc into two closed

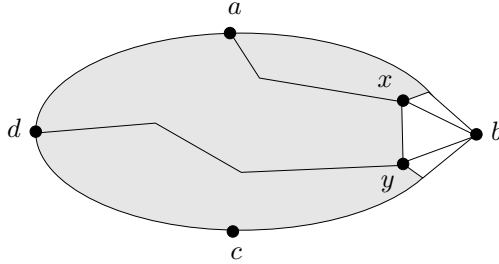


FIG. 8. Two disjoint paths, one in T_a and the other in T_d .

regions B and D , with b in B and d in D . Since $x \prec y$ and $x \in N_G(b)$, it follows from (i) that $T_a[a, x] \cap T_c[c, y] = \emptyset$, and since $y \prec z$ and $y \in N_G(b)$, it follows from (i) that $T_a[a, y] \cap T_c[c, z] = \emptyset$. So $T_a[a, x]$ lies in B and $T_c[c, z]$ lies in D . By planarity $T_a[a, x] \cap T_c[c, z] = \emptyset$. Therefore by (i), $x \prec z$.

- (7) $x, z \in N_G(b)$ and $y \in N_G(d)$. We have shown that either $x \prec z$ or $z \prec x$. Suppose for a contradiction that $z \prec x$. Then by (i), $T_a[a, z] \cap T_c[c, x] = \emptyset$, and by planarity, $T_a[a, x] \cap T_c[c, z] \neq \emptyset$. Since $y \prec z$ and $y \in N_G(d)$, $T_a[a, z] \cap T_c[c, y] \neq \emptyset$. But then, by planarity, $T_a[a, x] \cap T_c[c, y] \neq \emptyset$, which is a contradiction to (i) since $x \prec y$ and $x \in N_G(b)$. Therefore, $x \prec z$.
- (8) $x, z \in N_G(d)$ and $y \in N_G(b)$. We have shown that either $x \prec z$ or $z \prec x$. Suppose for a contradiction that $z \prec x$. Then by (ii), $T_a[a, z] \cap T_c[c, x] = \emptyset$ and $T_a[a, x] \cap T_c[c, z] \neq \emptyset$. Since $x \prec y$ and $x \in N_G(d)$, $T_a[a, y] \cap T_c[c, x] \neq \emptyset$. But then, by planarity, $T_a[a, y] \cap T_c[c, z] \neq \emptyset$, which is a contradiction to (i) since $y \prec z$ and $y \in N_G(b)$. Therefore, $x \prec z$.

Thus, \prec defines a total order on $(N_G(b) \cup N_G(d)) - \{a, c\}$. Hence, the required (T_a, T_c) -ordering exists.

Furthermore, this ordering can be found as follows. Let b_1, \dots, b_p be the (T_a, T_c) -ordering of $N_G(b) - \{a, c\}$, and let d_1, \dots, d_q be the (T_a, T_c) -ordering of $N_G(d) - \{a, c\}$. Both exist by Lemma 3.4. These sequences are ordered under \prec . We can decide in $O(|V(G)|)$ time whether $b_i \prec d_j$ or $d_j \prec b_i$ (by checking which of (i) or (ii) holds) for any pair b_i, d_j , $1 \leq i \leq p$, $1 \leq j \leq q$. Thus, using the so-called merge technique in [2], we can merge the two sequences to obtain a sequence ordered under \prec in $O(|V(G)|^2)$ time. \square

The last two lemmas of this section will also be needed in section 5. Figure 8 illustrates Lemma 3.6, and Figure 9 illustrates Lemma 3.7.

LEMMA 3.6. *Let (G, a, b, c, d) be a planar graph, and let $\{T_a, T_b, T_c, T_d\}$ be an independent spanning $\{a, b, c, d\}$ -system of G , where T_v is rooted at v for each $v \in \{a, b, c, d\}$. Assume that b has at least two neighbors in $V(G) - \{a, c\}$. Then for any (T_a, T_c) -ordered pair $x, y \in N_G(b) - \{a, c\}$, $T_a[a, x] \cap T_d[d, y] = \emptyset$.*

Proof. Take an embedding of G in a disc such that a, b, c, d occur in clockwise order on the boundary of the disc. Let $x, y \in N_G(b) - \{a, c\}$ such that x, y is (T_a, T_c) -ordered (see Figure 8). Hence, $T_a[a, x] \cap T_c[c, y] = \emptyset$. Since T_a and T_d are independent, $P := T_a[a, x] \cup T_d[d, x]$ is an a - d path in $G - \{a, c\}$, and P divides the disc into two closed regions B and C , with b in B . By planarity and since $T_a[a, x] \cap T_c[c, y] = \emptyset$, $T_d[d, y]$ lies in B . Then by planarity, $T_a[a, x] \cap T_d[d, y] = \emptyset$. \square

LEMMA 3.7. *Let (G, a, b, c, d) be a planar graph, and let $\{T_a, T_b, T_c, T_d\}$ be an independent spanning $\{a, b, c, d\}$ -system of G , where T_v is rooted at v for each $v \in$*

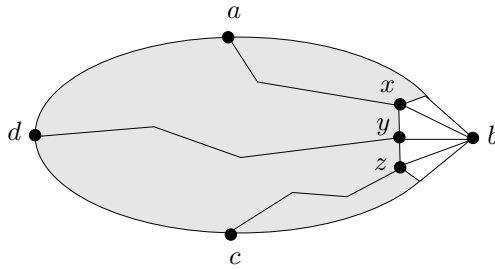


FIG. 9. Three disjoint paths contained in $T_a, T_c,$ and $T_d,$ respectively.

$\{a, b, c, d\}$. Assume that b has at least three neighbors in $V(G) - \{a, c\}$. Then for any (T_a, T_c) -ordered triple $x, y, z \in N_G(b) - \{a, c\}$, $T_a[a, x], T_d[d, y]$, and $T_c[c, z]$ are pairwise disjoint.

Proof. Take an embedding of G in a disc such that a, b, c, d occur in clockwise order on the boundary of the disc. Let $x, y, z \in N_G(b)$ such that x, y, z is (T_a, T_c) -ordered (see Figure 9). Hence, $T_a[a, x] \cap T_c[c, y] = \emptyset$ and $T_a[a, y] \cap T_c[c, z] = \emptyset$.

By Lemma 3.4, $T_a[a, x] \cap T_c[c, z] = \emptyset$. Hence, the path $P := T_d[d, y] + \{b, yb\}$ divides the disc into closed regions A and C , with $T_a[a, x]$ in A and $T_c[c, z]$ in C . By Lemma 3.6, $T_a[a, x] \cap T_d[d, y] = \emptyset$. By applying a mirror image version of Lemma 3.6, we can show that $T_d[d, y] \cap T_c[c, z] = \emptyset$. \square

4. Numberings. By Theorem 2.8, G has a nonseparating chain decomposition rooted at r . In this section we will combine this decomposition with Theorem 3.2 to produce a numbering of a subset of $V(G)$. This numbering will be used in the next section to construct four independent spanning trees rooted at r .

In the rest of this section we fix the following notation.

Notation 4.1. Let G be a 4-connected graph, and let $r \in V(G)$. Fix a nonseparating chain decomposition of G rooted at r , say $\mathcal{C} := (H_1, \dots, H_t)$, $t \geq 2$. Define the sequences $G_0, \bar{G}_1, \dots, \bar{G}_t$ as follows:

- (i) $G_0 := \bar{G}_t := (\{r\}, \emptyset)$.
- (ii) For $i = 1, \dots, t - 1$, $G_i := G[\bigcup_{j=1}^i I(H_j)]$ and $\bar{G}_i := G - (V(G_i) - \{r\})$.

Notation 4.2. Suppose that H_i ($1 \leq i \leq t$) is an up G_{i-1} -chain in G or a down G_{i-1} -chain in G . Let $H_i := v_0 B_1 v_1 B_2 v_2 \dots v_{k-1} B_k v_k$. For each 2-connected B_j there exist u_j, w_j (both on $V(G_{i-1})$ or both on $V(\bar{G}_i)$) such that $B_j - \{v_{j-1}, v_j\}$ is a component of $G - \{v_{j-1}, v_j, u_j, w_j\}$, and $(B_j^+, v_{j-1}, u_j, v_j, w_j)$ is planar, where $B_j^+ := G[V(B_j) \cup \{u_j, w_j\}] - u_j w_j$. We refer to each such B_j^+ as a *planar section* in C . The vertices v_{j-1}, v_j, u_j, w_j are the *terminals* of B_j^+ . See Figure 10 for an illustration. Note that the notation above depends on i . For the sake of clarity we will not make it explicit in the notation, but whenever we use this we will make clear which i we refer to. Furthermore, the algorithms we will describe deal with each H_i separately, and thus no confusion should arise.

DEFINITION 4.3. Suppose that H_i ($1 \leq i \leq t$) is a triangle G_{i-1} -chain in G . See Figure 4. Let $I(H_i) := \{v_1, v_2, v_3\}$, let $V(H_i) - I(H_i) := \{x_1, x_2, x_3\}$, and suppose that $x_j v_j \in E(G)$ for $j = 1, 2, 3$. We say that $v_j x_j$ ($j = 1, 2, 3$) are the *legs* of H_i .

DEFINITION 4.4. Let $D \subseteq V(G)$. A numbering of D is a function from D to $\{1, \dots, |D|\}$. Let g be a numbering of D , let v_1, \dots, v_k be a sequence of distinct vertices in $V(G) - D$, and let $v_0 \in D$. The extension g' of g to v_1, \dots, v_k from v_0 is defined as follows:

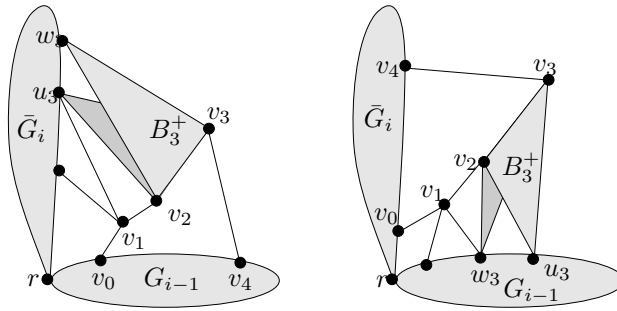


FIG. 10. v_2, v_3, u_3, w_3 are the terminals of B_3^+ .

- (i) for $1 \leq i \leq k$, let $g'(v_i) := g(v_0) + i$;
- (ii) for each $v \in D$ with $g(v) \leq g(v_0)$ let $g'(v) = g(v)$; and
- (iii) for each $v \in D$ with $g(v) > g(v_0)$ let $g'(v) := g(v) + k$.

Note that g' is a numbering of $D \cup \{v_1, \dots, v_k\}$. For convenience, if $D \subseteq V(G)$ and σ denotes a sequence v_1, \dots, v_k of vertices in $V(G) - D$, we let $D \cup \{\sigma\} := D \cup \{v_1, \dots, v_k\}$.

In order to compute the desired numberings g and f from a nonseparating chain decomposition, we need to find independent spanning systems in the planar sections in \mathcal{C} .

Assumption 4.5. For each planar section B_j^+ in \mathcal{C} , with terminals v_{j-1}, v_j, u_j, w_j , we compute an independent spanning $\{v_{j-1}, v_j, u_j, w_j\}$ -system of B_j^+ . By Theorem 3.2, such a system can be computed in $O(|V(B_j^+)| + |E(B_j^+)|)$ time. Since two distinct planar sections are edge-disjoint, the overall time consumed in this phase (for all planar sections in \mathcal{C}) is $O(|V(G)| + |E(G)|)$.

Next, we describe the algorithm for computing a numbering g of a subset of $V(G)$. It also computes a sequence $\{r\} = D_0 \subset D_1 \subset \dots \subset D_{t-1}$ of subsets of $V(G)$ such that for $i = 1, \dots, t$, $N_G(H_i) \cap V(G_{i-1}) \subseteq D_{i-1}$. When the algorithm stops, g is a numbering of D_{t-1} . We note that keeping track of this sequence is not necessary for computing g , but its inclusion will make proofs easier in section 6.

ALGORITHM NUMBERING g .

Description. The algorithm executes $t - 1$ iterations, where t is the number of chains in \mathcal{C} . At the beginning of the first iteration, we have $i = 1$, $D_0 := \{r\}$, and $g(r) := 1$. At the beginning of each iteration, we have an integer i with $1 \leq i \leq t - 1$, a subset $D_{i-1} \subseteq V(G_{i-1})$ such that $N_G(H_i) \cap V(G_{i-1}) \subseteq D_{i-1}$, and a numbering g of D_{i-1} .

Each iteration consists of the following: update g and define D_i according to the following cases (depending on the type of H_i), and, if $i < t - 1$, then set $i \leftarrow i + 1$ and start a new iteration.

Case 1. H_i is an elementary G_{i-1} -chain in G .

Let $H_i := v_0 B_1 v_1 B_2 v_2$, and assume that v_0, v_2 are labeled so that $g(v_0) < g(v_2)$.

Extend g to v_1 from v_0 , and let $D_i := D_{i-1} \cup \{v_1\}$.

Case 2. $i = 1$, or H_i is an up G_{i-1} -chain in G but not an elementary G_{i-1} -chain.

Let $H_i := v_0 B_1 v_1 \dots v_{k-1} B_k v_k$, and suppose that v_0, \dots, v_k and B_1, \dots, B_k are labeled so that $v_0 = v_k = r$ when $i = 1$ and $g(v_0) < g(v_k)$ when $i \neq 1$. For each 2-connected B_j , let u_j, w_j denote the terminals of B_j^+ other than v_{j-1}, v_j . Let $T_{v_{j-1}}^j, T_{v_j}^j$ denote the trees rooted, respectively, at v_{j-1}, v_j in the independent spanning $\{v_{j-1}, v_j, u_j, w_j\}$ -system of B_j^+ computed in Assumption 4.5.

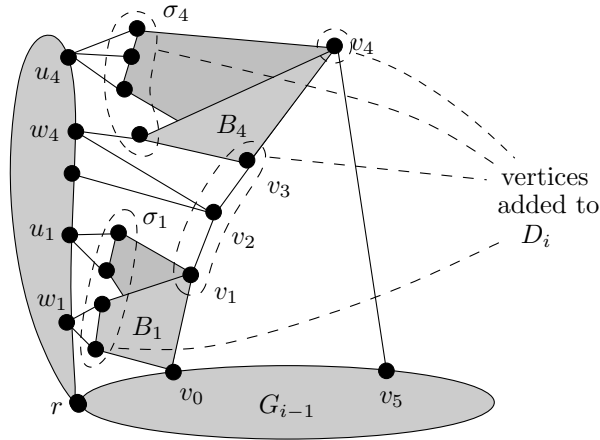


FIG. 11. Extending the numbering g to an up G_{i-1} -chain.

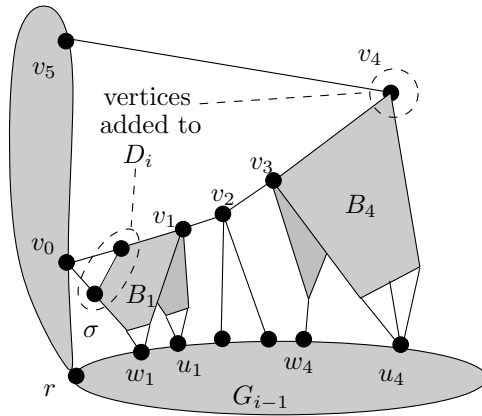


FIG. 12. Extending the numbering g to a down G_{i-1} -chain.

For each $j = 1, \dots, k$, compute a sequence σ_j as follows. If B_j is 2-connected, then let σ_j be a $(T_{v_{j-1}}^j, T_{v_j}^j)$ -ordering of $N_{B_j^+}(\{u_j, w_j\}) - \{v_{j-1}, v_j\}$ (the existence of this ordering is guaranteed by Lemma 3.5). If B_j is trivial, then let σ_j denote the empty sequence.

Extend g to $\sigma := \sigma_1, v_1, \sigma_2, v_2, \dots, v_{k-1}, \sigma_k$ from v_0 , and let $D_i := D_{i-1} \cup \{\sigma\}$. See Figure 11 for an illustration.

Case 3. H_i is a down G_{i-1} -chain in G but not an elementary G_{i-1} -chain.

Let $H_i := v_0 B_1 v_1 \dots v_{k-1} B_k v_k$. For each 2-connected block B_j let u_j, w_j denote the terminals of B_j^+ other than v_{j-1}, v_j with $g(u_j) < g(w_j)$. Let $T_{u_j}^j, T_{w_j}^j$ denote trees rooted, respectively, at u_j, w_j in the independent spanning $\{v_{j-1}, v_j, u_j, w_j\}$ -system of B_j^+ computed in Assumption 4.5.

Let $D_i := D_{i-1} \cup N_{B_1}(v_0) \cup N_{B_k}(v_k)$. See Figure 12 for an illustration. Extend g according to the following three subcases.

Subcase 3.1. $k = 1$ (thus, B_1 is 2-connected).

Let σ denote a $(T_{u_1}^1, T_{w_1}^1)$ -ordering of $N_{B_1^+}(\{v_0, v_1\}) - \{u_1, w_1\} = N_{B_1}(v_0) \cup N_{B_k}(v_k)$ (the existence of this ordering is guaranteed by Lemma 3.5). Extend g to σ from u_1 .

Subcase 3.2. $k = 2$, and B_1 or B_2 is trivial.

Note that since H_i is not an elementary G_{i-1} -chain, B_1 or B_2 is nontrivial.

Assume then (renaming B_1 and B_2 if necessary) that B_1 is 2-connected and B_2 is trivial. Extend g according to the following cases.

- (i) v_1 has no neighbor in $V(G_{i-1})$. Let q_1, q_2, q_3 be neighbors of v_1 in B_1 (they exist since G is 4-connected), and assume that q_1, q_2, q_3 is $(T_{u_1}^1, T_{w_1}^1)$ -ordered (this is possible by Lemma 3.4). By Lemma 3.7, $T_{u_1}^1[u_1, q_1], T_{v_0}^1[v_0, q_2]$, and $T_{w_1}^1[w_1, q_3]$ are disjoint. Let $H := B_1^+ \cup B_2$ (note that $H - \{v_0, v_2, u_1, w_1\}$ is a component of $G - \{v_0, v_2, u_1, w_1\}$). Then (H, v_0, u_1, v_2, w_1) is planar, and $\{T_{v_0}^1 + \{v_1, v_1q_2\}, T_{v_1}^1 + \{v_2, v_1v_2\}, T_{u_1}^1 + \{v_1, v_1q_1\}, T_{w_1}^1 + \{v_1, v_1q_3\}\}$ forms an independent spanning $\{v_0, u_1, v_2, w_1\}$ -system of H .

Let σ denote a $(T_{u_1}^1 + \{v_1, v_1q_1\}, T_{w_1}^1 + \{v_1, v_1q_3\})$ -ordering of $N_H(\{v_0, v_2\}) - \{u_1, w_1\}$ (the existence of this ordering is guaranteed by Lemma 3.5). Extend g to σ from u_1 .

Comment: we also keep track of q_1, q_2, q_3 for the construction of the independent spanning trees.

- v_1 has a neighbor in $V(G_{i-1})$. Let $x \in N_G(v_1) \cap V(G_{i-1})$ with $g(x)$ minimum, and let σ denote a $(T_{u_1}^1, T_{w_1}^1)$ -ordering of $N_{B_1^+}(v_0) - \{u_1, w_1\}$ (the existence of this ordering is guaranteed by Lemma 3.4). If $g(x) > g(u_1)$, then extend g to σ, v_1 from u_1 , where σ, v_1 is the sequence obtained from σ by adding v_1 at the end. If $g(x) \leq g(u_1)$, then extend g to v_1, σ from x , where v_1, σ is the sequence obtained from σ by adding v_1 in the front.

Subcase 3.3. $k \geq 3$, or $k = 2$ and both B_1, B_2 are 2-connected.

Extend g to $N_{B_1}(v_0)$ according to the following cases.

- B_1 is 2-connected. Let σ denote a $(T_{u_1}^1, T_{w_1}^1)$ -ordering of $N_{B_1^+}(v_0) - \{u_1, w_1\} = N_{B_1}(v_0)$ (the existence of this ordering is guaranteed by Lemma 3.4). Extend g to σ from u_1 .
- Both B_1 and B_2 are trivial. Let $x \in N_G(v_1) \cap V(G_{i-1})$ with $g(x)$ minimum. Extend g to v_1 from x .
- B_1 is trivial and B_2 is 2-connected.
 - If v_1 has no neighbor in $V(G_{i-1})$, extend g to v_1 from u_2 .
 - If v_1 has a neighbor in $V(G_{i-1})$, let $x \in N_G(v_1) \cap V(G_{i-1})$ with $g(x)$ minimum. If $g(x) > g(u_2)$, then extend g to v_1 from u_2 . If $g(x) \leq g(u_2)$, then extend g to v_1 from x .

Extend (the resulting) g to $N_{B_k}(v_k)$ according to the following cases.

- B_k is 2-connected. Let σ be a $(T_{u_k}^k, T_{w_k}^k)$ -ordering of $N_{B_k^+}(v_k) - \{u_k, w_k\} = N_{B_k}(v_k)$ (the existence of this ordering is guaranteed by Lemma 3.4). Extend g to σ from u_k .
- Both B_k and B_{k-1} are trivial. Let $x \in N_G(v_{k-1}) \cap V(G_{i-1})$ with $g(x)$ minimum. Extend g to v_{k-1} from x .
- B_k is trivial and B_{k-1} is 2-connected.
 - If v_{k-1} has no neighbor in $V(G_{i-1})$, extend g to v_{k-1} from u_{k-1} .
 - If v_{k-1} has a neighbor in $V(G_{i-1})$, let $x \in N_G(v_{k-1}) \cap V(G_{i-1})$ with $g(x)$ minimum. If $g(x) > g(u_{k-1})$, then extend g to v_{k-1} from u_{k-1} . If $g(x) \leq g(u_{k-1})$, then extend g to v_{k-1} from x .

Case 4. H_i is a triangle G_{i-1} -chain in G .

Let $I(H_i) := \{v_1, v_2, v_3\}$, and let $v_j x_j$ ($j = 1, 2, 3$) be the legs of H_i . Suppose that v_1, v_2, v_3 are labeled so that $g(x_1) < g(x_2) < g(x_3)$. Let $D_i := D_{i-1} \cup \{v_1, v_2, v_3\}$. Extend g to v_1, v_2, v_3 from x_2 .

This concludes the description of the algorithm for computing g .

LEMMA 4.6. *Algorithm Numbering g runs in $O(|V(G)|^3)$ time.*

Proof. Observe that at the i th iteration, Algorithm Numbering g extends the current numbering g^i to a sequence σ from a previously numbered vertex $v \in D_{i-1}$. Clearly, given g^i, σ , and v , this extension can be computed in $O(|V(G)|)$ time. We now analyze the time spent at each iteration of Algorithm Numbering g according to Cases 1–4. We use the same notation as in the algorithm.

If Case 1 occurs (H_i is an elementary G_{i-1} -chain in G), then Algorithm Numbering g extends g to v_1 . This can be done in $O(|V(G)|)$ time.

If Case 2 occurs (H_i is an up G_{i-1} -chain but not an elementary G_{i-1} -chain), then Algorithm Numbering g computes sequences $\sigma_1, \dots, \sigma_k$, where σ_j denotes the empty sequence when B_j is trivial, and σ_j is a $(T_{v_{j-1}}^j, T_{v_j}^j)$ -ordering of $N_{B_j^+}(\{u_j, w_j\}) - \{v_{j-1}, v_j\}$ when B_j is 2-connected. In the latter case, by Lemma 3.5 the sequence σ_j can be computed in $O(|V(B_j^+)|^2)$ time. Thus, the algorithm spends $O(|V(G)|^2)$ time to compute $\sigma_1, \dots, \sigma_k$. After that, the algorithm extends g to $v_0, \sigma_1, v_1, \dots, v_{k-1}, \sigma_k, v_k$, which can be done in $O(|V(G)|)$ time. Therefore, the algorithm spends $O(|V(G)|^2)$ time if Case 2 occurs.

If Case 3 occurs (H_i is a down G_{i-1} -chain but not an elementary G_{i-1} -chain), then Algorithm Numbering g considers three cases.

- If Subcase 3.1 occurs ($k = 1$), then the algorithm computes a $(T_{u_1}^1, T_{w_1}^1)$ -ordering σ of $N_{B_1^+}(\{v_0, v_1\}) - \{u_1, w_1\}$ and extends g to σ from u_1 . The sequence σ can be computed in $O(|V(G)|^2)$ time by Lemma 3.5, and the extension of g can be computed in $O(|V(G)|)$ time, resulting in $O(|V(G)|^2)$ time for this iteration.
- If Subcase 3.2 occurs ($k = 2$, and B_1 or B_2 is trivial), then the algorithm considers two subcases, according to whether or not v_1 has a neighbor in $V(G_{i-1})$.
 - If v_1 has no neighbor in $V(G_{i-1})$, the algorithm chooses neighbors q_1, q_2, q_3 of v_1 in B_1 and computes a $(T_{u_1}^1 + \{v_1, v_1 q_1\}, T_{w_1}^1 + \{v_1, v_1 q_3\})$ -ordering σ of $N_{B_1}(v_0) \cup N_{B_k}(v_k) = N_H(\{v_0, v_2\}) - \{u_1, w_1\}$ as in Subcase 3.1 and extends g to σ from u_1 . Thus, the algorithm spends $O(|V(G)|^2)$ time in this case.
 - If v_1 has a neighbor in $V(G_{i-1})$, then the algorithm computes a $(T_{u_1}^1, T_{w_1}^1)$ -ordering σ of $N_{B_1^+}(v_0) - \{u_1, w_1\}$, and it performs an extension on g . The sequence σ can be computed in $O(|V(G)|)$ time by Lemma 3.4, and the extension can be computed in $O(|V(G)|)$ time. Thus, the algorithm spends $O(|V(G)|)$ time in this case.
- If Subcase 3.3 occurs ($k \geq 3$, or $k = 2$ and both B_1, B_2 are 2-connected), then the algorithm extends g to $N_{B_1}(v_0)$ and extends g to $N_{B_k}(v_k)$. The algorithm may need to compute a $(T_{u_1}^1, T_{w_1}^1)$ -ordering of $N_{B_1^+}(v_0) - \{u_1, w_1\}$ and a $(T_{u_k}^k, T_{w_k}^k)$ -ordering of $N_{B_k^+}(v_k) - \{u_k, w_k\}$, but both can be done in $O(|V(G)|)$ time by Lemma 3.4. It is not hard to check that the algorithm spends $O(|V(G)|)$ time in this case.

If Case 4 occurs (H_i is a triangle chain), then Algorithm Numbering g extends g to v_1, v_2, v_3 . This can be done in $O(|V(G)|)$ time.

From the analysis above, it follows that Algorithm Numbering g spends $O(|V(G)|^2)$ time in each iteration. Since the number of iterations is $t < n \rightarrow t < |V(G)|$, the numbering g can be computed in $O(|V(G)|^3)$ time. \square

Note that the extension operation does not affect the order of the vertices previously numbered, although their actual g values may have changed. Thus, at each iteration the algorithm orders the vertices in $D_i - D_{i-1}$ without affecting the order of the vertices in D_{i-1} . In fact, it does not affect the order of the vertices in D_j for every $1 \leq j \leq i - 1$.

The numbering g will be used to construct *two* independent spanning trees rooted at r from $\mathcal{C} = (H_1, \dots, H_t)$ in order from H_1 to H_t . For constructing the other two spanning trees we compute a numbering f by examining the chains of \mathcal{C} in reverse order.

The algorithm for computing f is analogous to Algorithm Numbering g when it deals with an up G_{i-1} -chain or a down G_{i-1} -chain or elementary G_{i-1} -chain. The differences appear when it deals with a triangle G_{i-1} -chain. The algorithm also computes a sequence $\{r\} = D'_{t+1} \subset D'_t \subset \dots \subset D'_2$ of subsets of $V(G)$ such that for $t \geq i \geq 1$, $N_G(H_i) \cap V(\bar{G}_i) \subseteq D'_{i+1}$.

ALGORITHM NUMBERING f .

Description. The algorithm executes $t - 1$ iterations, where t is the number of chains in $\mathcal{C}' := (H_1, \dots, H_t)$. At the beginning of the first iteration, we have $i = t$, $D'_{t+1} := \{r\}$, and $f(r) := 1$. At the beginning of each iteration, we have an integer i with $t \geq i \geq 2$, a subset $D'_{i+1} \subseteq V(\bar{G}_i)$ such that $N_G(H_i) \cap V(\bar{G}_i) \subseteq D'_{i+1}$, and a numbering f of D'_{i+1} .

Each iteration consists of the following: update f and define D'_i according to the following cases (depending on the chain type of H_i), and, if $i > 2$, then set $i \leftarrow i - 1$ and start a new iteration.

Case 1. H_i is an elementary G_{i-1} -chain in G .

Let $H_i := v_0 B_1 v_1 B_2 v_2$, and let v'_0, v'_2 be neighbors of v_1 in $V(\bar{G}_i)$ with $f(v'_0) < f(v'_2)$. Extend f to v_1 from v'_0 , and let $D'_i := D'_{i+1} \cup \{v_1\}$.

Case 2. $i = t$, or H_i is a down G_{i-1} -chain in G but not an elementary G_{i-1} -chain.

Let $H_i := v_0 B_1 v_1 \dots v_{k-1} B_k v_k$, and suppose that v_0, \dots, v_k and B_1, \dots, B_k are labeled so that $v_0 = v_k = r$ when $i = t$ and $f(v_0) < f(v_k)$ when $i \neq t$. For each 2-connected B_j , let u_j, w_j be the terminals of B_j^+ other than v_{j-1}, v_j . Let $T^j_{v_{j-1}}, T^j_{v_j}$ denote the trees rooted, respectively, at v_{j-1}, v_j in the independent spanning $\{v_{j-1}, v_j, u_j, w_j\}$ -system of B_j^+ computed in Assumption 4.5.

For each $j = 1, \dots, k$ compute a sequence σ_j as follows. If B_j is 2-connected, then let σ_j be a $(T^j_{v_{j-1}}, T^j_{v_j})$ -ordering of $N_{B_j^+}(\{u_j, w_j\}) - \{v_{j-1}, v_j\}$ (the existence of this ordering is guaranteed by Lemma 3.5). If B_j is trivial, then let σ_j denote the empty sequence.

Extend f to $\sigma := \sigma_1, v_1, \sigma_2, v_2, \dots, v_{k-1}, \sigma_k$ from v_0 , and let $D'_i := D'_{i+1} \cup \{\sigma\}$. See Figure 13 for an illustration.

Case 3. H_i is an up G_{i-1} -chain in G but not an elementary G_{i-1} -chain in G .

Let $H_i := v_0 B_1 v_1 \dots v_{k-1} B_k v_k$. For each 2-connected B_j , let u_j, w_j denote the terminals of B_j^+ other than v_{j-1}, v_j , with $f(u_j) < f(w_j)$. Let $T^j_{u_j}, T^j_{w_j}$ denote the trees rooted, respectively, at u_j, w_j in the independent spanning $\{v_{j-1}, v_j, u_j, w_j\}$ -system of B_j^+ computed in Assumption 4.5.

Let $D'_i := D'_{i+1} \cup N_{B_1}(v_0) \cup N_{B_k}(v_k)$. See Figure 14 for an illustration. Extend f according to the following three subcases.

Subcase 3.1. $k = 1$ (thus, B_1 is 2-connected).

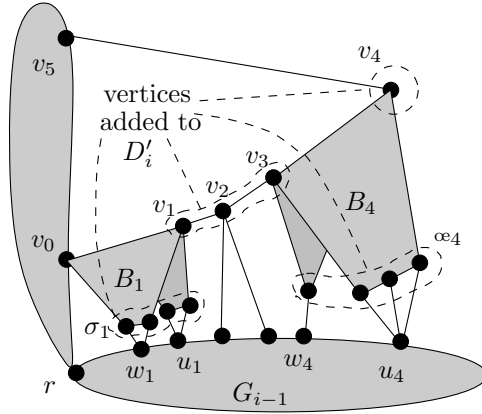


FIG. 13. Extending the numbering f to a down G_{i-1} -chain.

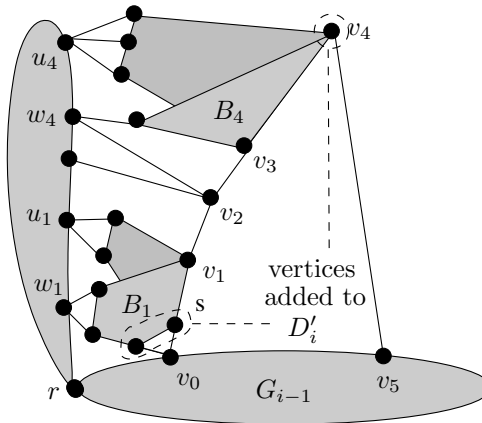


FIG. 14. Extending the numbering f to an up G_{i-1} -chain.

Let σ denote a $(T_{u_1}^1, T_{w_1}^1)$ -ordering of $N_{B_1^+}(\{v_0, v_1\}) - \{u_1, w_1\} = N_{B_1}(v_0) \cup N_{B_k}(v_k)$ (the existence of this ordering is guaranteed by Lemma 3.5). Extend f to σ from u_1 .

Subcase 3.2. $k = 2$, and B_1 or B_2 is trivial.

Note that since H_i is not an elementary chain, B_1 or B_2 is nontrivial.

Assume then (renaming B_1 and B_2 if necessary) that B_1 is 2-connected and B_2 is trivial. Extend f according to the following cases.

- v_1 has no neighbor in $V(\bar{G}_i)$. Let q_1, q_2, q_3 be distinct neighbors of v_1 in B_1 (they exist since G is 4-connected), and assume that q_1, q_2, q_3 is $(T_{u_1}^1, T_{w_1}^1)$ -ordered (this is possible by Lemma 3.4). By Lemma 3.7, $T_{u_1}^1[u_1, q_1], T_{v_0}^1[v_0, q_2]$, and $T_{w_1}^1[w_1, q_3]$ are disjoint. Let $H := B_1^+ \cup B_2$. Note that $H - \{v_0, v_2, u_1, w_1\}$ is a component of $G - \{v_0, v_2, u_1, w_1\}$, (H, v_0, u_1, v_2, w_1) is planar, and $\{T_{v_0}^1 + \{v_1, v_1q_2\}, T_{v_1}^1 + \{v_2, v_1v_2\}, T_{u_1}^1 + \{v_1, v_1q_1\}, T_{w_1}^1 + \{v_1, v_1q_3\}\}$ is an independent spanning $\{v_0, v_2, u_1, w_1\}$ -system of H .
Let σ denote a $(T_{u_1}^1 + \{v_1, v_1q_1\}, T_{w_1}^1 + \{v_1, v_1q_3\})$ -ordering of $N_H(\{v_0, v_2\}) - \{u_1, w_1\} = N_{B_1}(v_0) \cup N_{B_k}(v_k)$ (the existence of this ordering is guaranteed by Lemma 3.5). Extend f to σ from u_1 .

Comment: we also keep track of q_1, q_2, q_3 for the construction of the independent spanning trees.

- v_1 has a neighbor in $V(\bar{G}_i)$. Let $x \in N_G(v_1) \cap V(\bar{G}_i)$ with $f(x)$ minimum, and let σ denote a $(T_{u_1}^1, T_{w_1}^1)$ -ordering of $N_{B_1}(v_0) = N_{B_1^+}(v_0) - \{u_1, w_1\}$ (the existence of this ordering is guaranteed by Lemma 3.4). If $f(x) > f(u_1)$, then extend f to σ, v_1 from u_1 . If $f(x) \leq f(u_1)$, then extend f to v_1, σ from x .

Subcase 3.3. $k \geq 3$, or $k = 2$ and both B_1, B_2 are 2-connected.

Extend f to $N_{B_1}(v_0)$ according to the following cases.

- B_1 is 2-connected. Let σ denote a $(T_{u_1}^1, T_{w_1}^1)$ -ordering of $N_{B_1}(v_0) = N_{B_1^+}(v_0) - \{u_1, w_1\}$ (the existence of this ordering is guaranteed by Lemma 3.4). Extend f to σ from u_1 .
- Both B_1 and B_2 are trivial. Let $x \in N_G(v_1) \cap V(\bar{G}_i)$ with $f(x)$ minimum. Extend f to v_1 from x .
- B_1 is trivial and B_2 is 2-connected.
 - If v_1 has no neighbor in $V(\bar{G}_i)$, extend f to v_1 from u_2 .
 - If v_1 has a neighbor in $V(\bar{G}_i)$, let $x \in N_G(v_1) \cap V(\bar{G}_i)$ with $f(x)$ minimum. If $f(x) > f(u_2)$, then extend f to v_1 from u_2 . If $f(x) \leq f(u_2)$, then extend f to v_1 from x .

Extend (the resulting) f to $N_{B_k}(v_k)$ according to the following cases.

- B_k is 2-connected. Let σ denote a $(T_{u_k}^k, T_{w_k}^k)$ -ordering of $N_{B_k}(v_k) = N_{B_k^+}(v_k) - \{u_k, w_k\}$ (the existence of this ordering is guaranteed by Lemma 3.4). Extend f to σ from u_k .
- Both B_k and B_{k-1} are trivial. Let $x \in N_G(v_{k-1}) \cap V(\bar{G}_i)$ with $f(x)$ minimum. Extend f to v_{k-1} from x .
- B_k is trivial and B_{k-1} is 2-connected.
 - If v_{k-1} has no neighbor in $V(\bar{G}_i)$, extend f to v_{k-1} from u_{k-1} .
 - If v_{k-1} has a neighbor in $V(\bar{G}_i)$, let $x \in N_G(v_{k-1}) \cap V(\bar{G}_i)$ with $f(x)$ minimum. If $f(x) > f(u_{k-1})$, then extend f to v_{k-1} from u_{k-1} . If $f(x) \leq f(u_{k-1})$, then extend f to v_{k-1} from x .

Case 4. H_i is a triangle G_{i-1} -chain in G .

Let $I(H_i) := \{v_1, v_2, v_3\}$, let $v_j x_j$ ($j = 1, 2, 3$) be the legs of H_i , and let $y_1, y_2, y_3 \in V(\bar{G}_i)$ such that $y_1 v_1, y_2 v_2, y_3 v_3 \in E(G)$. Assume that v_1, v_2, v_3 are labeled so that $g(x_1) < g(x_2) < g(x_3)$. Let $D'_i := D'_{i+1} \cup \{v_1, v_2, v_3\}$.

- If $f(y_1) < f(y_2)$ and $f(y_1) < f(y_3)$, then extend f to v_1, v_2, v_3 from y_1 .
- If $f(y_2) < f(y_1)$ and $f(y_2) < f(y_3)$, then extend f to v_2, v_1, v_3 from y_2 .
- If $f(y_3) < f(y_1) < f(y_2)$, then extend f to v_3 from y_3 and extend (the resulting) f to v_1, v_2 from y_1 .
- If $f(y_3) < f(y_2) < f(y_1)$, then extend f to v_3 from y_3 and extend (the resulting) f to v_2, v_1 from y_2 .

This concludes the description of the algorithm for computing f . The proof of the next lemma is similar to the proof of Lemma 4.6, and we omit it.

LEMMA 4.7. *Algorithm Numbering f runs in $O(|V(G)|^3)$ time.*

5. Construction of spanning trees. We now describe how to use Theorem 3.2 and the two numberings of the last section to produce four independent spanning trees. This will follow from Algorithm Trees. The proof of its correction and analysis of its complexity will be given in the next section.

ALGORITHM TREES.

Description. Let G be a 4-connected graph, let $r \in V(G)$, and let $\mathcal{C} = (H_1, \dots, H_t)$ be a nonseparating chain decomposition of G rooted at r . Let $G_0 = \bar{G}_t = (\{r\}, \emptyset)$,

and for $1 \leq i \leq t - 1$, let $G_i := G[\bigcup_{j=1}^i I(H_j)]$ and $\bar{G}_i = G - (V(G_i) - \{r\})$. The algorithm executes t iterations, where t is the number of chains in \mathcal{C} . At the first iteration, we have $i = 1$ and $T_1 = T_2 = T_3 = T_4 = G_0$. At the beginning of each iteration, we have an integer i with $1 \leq i \leq t$, spanning trees T_1, T_2 in G_{i-1} and spanning forests T_3, T_4 in $G_{i-1} - r$.

Each iteration consists of the following: update T_1, T_2, T_3, T_4 by adding certain vertices and edges of H_i to T_1, T_2, T_3, T_4 according to the following four cases (depending on the type of H_i), and, if $i < t$, then set $i \leftarrow i + 1$ and start a new iteration. After t iterations, T_1, T_2, T_3, T_4 will be independent spanning trees in G rooted at r .

Case 1. H_i is an elementary G_{i-1} -chain in G .

Let $H_i := v_0 B_1 v_1 B_2 v_2$ with $g(v_0) < g(v_2)$. Let v'_0, v'_2 be neighbors of v_1 in $V(\bar{G}_i)$ with $f(v'_0) < f(v'_2)$.

Set $T_1 \leftarrow T_1 + \{v_1, v_0 v_1\}, T_2 \leftarrow T_2 + \{v_1, v_1 v_2\}, T_3 \leftarrow T_3 + \{v'_0, v_1, v'_0 v_1\}$, and $T_4 \leftarrow T_4 + \{v'_2, v_1, v'_2 v_1\}$.

Case 2. $i = 1$, or H_i is an up G_{i-1} -chain in G but not an elementary G_{i-1} -chain in G .

Let $H_i := v_0 B_1 v_1 \dots v_{k-1} B_k v_k$, with $v_0 = v_k = r$ when $i = 1$, and $g(v_0) < g(v_k)$ when $i \neq 1$.

For each 2-connected block B_j , let u_j, w_j denote the terminals of B_j^+ other than v_{j-1}, v_j with $f(u_j) < f(w_j)$, and let $T_{v_{j-1}}^j, T_{v_j}^j, T_{u_j}^j, T_{w_j}^j$ denote the trees rooted, respectively, at v_{j-1}, v_j, u_j, w_j in the independent spanning $\{v_{j-1}, v_j, u_j, w_j\}$ -system of B_j^+ computed in Assumption 4.5.

Let $J := \{j : 1 \leq j \leq k, B_j \text{ is 2-connected}\}$, and let $\bar{J} := \{1, \dots, k\} - J$.

First, set

$$\begin{aligned}
 T_1 &\leftarrow T_1 \cup \left(\bigcup_{j \in \bar{J} - \{k\}} B_j \right) \cup \left(\bigcup_{j \in J} T_{v_{j-1}}^j \right), \\
 T_2 &\leftarrow T_2 \cup \left(\bigcup_{j \in \bar{J} - \{1\}} B_j \right) \cup \left(\bigcup_{j \in J} T_{v_j}^j \right), \\
 T_3 &\leftarrow T_3 \cup \left(\bigcup_{j \in J} T_{u_j}^j \right), \text{ and} \\
 T_4 &\leftarrow T_4 \cup \left(\bigcup_{j \in J} T_{w_j}^j \right).
 \end{aligned}$$

Now for each $j = 1, \dots, k - 1$ add v_j and edges incident to v_j to T_1, T_2, T_3, T_4 according to the following cases (at this stage, $v_0, v_k \notin V(T_3 \cup T_4)$).

Subcase 2.1. B_j and B_{j+1} are trivial.

Let p_3, p_4 be neighbors of v_j in $V(\bar{G}_i)$ with $f(p_3)$ minimum (hence $f(p_3) < f(p_4)$). Set $T_3 \leftarrow T_3 + \{v_j, p_3, v_j p_3\}$ and $T_4 \leftarrow T_4 + \{v_j, p_4, v_j p_4\}$.

Subcase 2.2. B_j is 2-connected and B_{j+1} is trivial.

- If v_j has no neighbor in $V(\bar{G}_i)$, then let p_1, p_3, p_4 be neighbors of v_j in B_j (they exist since G is 4-connected), and assume that p_3, p_1, p_4 is $(T_{u_j}^j, T_{w_j}^j)$ -ordered (this is possible by Lemma 3.4). By Lemma 3.7, $T_{u_j}^j[u_j, p_3], T_{v_{j-1}}^j[v_{j-1}, p_1]$,

and $T_{w_j}^j[w_j, p_4]$ are disjoint. If $k = 2$, then we also require that p_3, p_1, p_4 be the vertices q_1, q_2, q_3 , respectively, chosen in Subcase 3.2 of Algorithm Numbering f .

Set $T_1 \leftarrow T_1 + \{v_j, v_j p_1\}$, $T_3 \leftarrow T_3 + \{v_j, v_j p_3\}$, and $T_4 \leftarrow T_4 + \{v_j, v_j p_4\}$.

- If v_j has a neighbor in $V(\bar{G}_i)$, then let $x \in N_G(v_j) \cap V(\bar{G}_i)$ with $f(x)$ minimum.
 - If $f(x) > f(u_j)$, then let p_1, p_3 be neighbors of v_j in B_j such that the paths $T_{v_{j-1}}^j[v_{j-1}, p_1]$ and $T_{u_j}^j[u_j, p_3]$ are disjoint (they exist by Lemma 3.6).
Set $T_1 \leftarrow T_1 + \{v_j, v_j p_1\}$, $T_3 \leftarrow T_3 + \{v_j, v_j p_3\}$, and $T_4 \leftarrow T_4 + \{v_j, x, v_j x\}$.
 - If $f(x) \leq f(u_j)$, then let p_1, p_4 be neighbors of v_j in B_j such that the paths $T_{v_{j-1}}^j[v_{j-1}, p_1]$ and $T_{w_j}^j[w_j, p_4]$ are disjoint (they exist by Lemma 3.6).
Set $T_1 \leftarrow T_1 + \{v_j, v_j p_1\}$, $T_3 \leftarrow T_3 + \{v_j, x, v_j x\}$, and $T_4 \leftarrow T_4 + \{v_j, v_j p_4\}$.

Subcase 2.3. B_j is trivial and B_{j+1} is 2-connected.

- If v_j has no neighbor in $V(\bar{G}_i)$, then let p_2, p_3, p_4 be neighbors of v_j in B_{j+1} (they exist since G is 4-connected), and assume that p_3, p_2, p_4 is $(T_{u_{j+1}}^{j+1}, T_{w_{j+1}}^{j+1})$ -ordered (this is possible by Lemma 3.4). By Lemma 3.7, $T_{u_{j+1}}^{j+1}[u_{j+1}, p_3]$, $T_{v_{j+1}}^{j+1}[v_{j+1}, p_2]$, and $T_{w_{j+1}}^{j+1}[w_{j+1}, p_4]$ are disjoint. If $k = 2$, then we also require that p_3, p_2, p_4 be the vertices q_1, q_2, q_3 , respectively, chosen in Subcase 3.2 of Algorithm Numbering f .
Set $T_2 \leftarrow T_2 + \{v_j, v_j p_2\}$, $T_3 \leftarrow T_3 + \{v_j, v_j p_3\}$, and $T_4 \leftarrow T_4 + \{v_j, v_j p_4\}$.
- If v_j has a neighbor in $V(\bar{G}_i)$, then let $x \in N_G(v_j) \cap V(\bar{G}_i)$ with $f(x)$ minimum.
 - If $f(x) > f(u_{j+1})$, then let p_2, p_3 be neighbors of v_j in B_{j+1} such that the paths $T_{v_{j+1}}^{j+1}[v_{j+1}, p_2]$ and $T_{u_{j+1}}^{j+1}[u_{j+1}, p_3]$ are disjoint (they exist by Lemma 3.6).
Set $T_2 \leftarrow T_2 + \{v_j, v_j p_2\}$, $T_3 \leftarrow T_3 + \{v_j, v_j p_3\}$, and $T_4 \leftarrow T_4 + \{v_j, x, v_j x\}$.
 - If $f(x) \leq f(u_{j+1})$, then let p_2, p_4 be neighbors of v_j in B_{j+1} such that the paths $T_{v_{j+1}}^{j+1}[v_{j+1}, p_2]$ and $T_{w_{j+1}}^{j+1}[w_{j+1}, p_4]$ are disjoint (they exist by Lemma 3.6).
Set $T_2 \leftarrow T_2 + \{v_j, v_j p_2\}$, $T_3 \leftarrow T_3 + \{v_j, x, v_j x\}$, and $T_4 \leftarrow T_4 + \{v_j, v_j p_4\}$.

Subcase 2.4. B_j and B_{j+1} are 2-connected.

Note that $f(u_j) < f(w_{j+1})$ or $f(u_{j+1}) < f(w_j)$.

- If $f(u_j) < f(w_{j+1})$, then let p_1, p_3 be neighbors of v_j in B_j such that the paths $T_{v_{j-1}}^j[v_{j-1}, p_1]$ and $T_{u_j}^j[u_j, p_3]$ are disjoint (they exist by Lemma 3.6), and let p_2, p_4 be neighbors of v_j in B_{j+1} such that the paths $T_{v_{j+1}}^{j+1}[v_{j+1}, p_2]$ and $T_{w_{j+1}}^{j+1}[w_{j+1}, p_4]$ are disjoint (they exist by Lemma 3.6).
Set $T_1 \leftarrow T_1 + \{v_j, v_j p_1\}$, $T_2 \leftarrow T_2 + \{v_j, v_j p_2\}$, $T_3 \leftarrow T_3 + \{v_j, v_j p_3\}$, and $T_4 \leftarrow T_4 + \{v_j, v_j p_4\}$.
- If $f(u_j) \geq f(w_{j+1})$, then $f(u_{j+1}) < f(w_j)$. Let p_1, p_4 be neighbors of v_j in B_j such that the paths $T_{v_{j-1}}^j[v_{j-1}, p_1]$ and $T_{w_j}^j[w_j, p_4]$ are disjoint (they exist by Lemma 3.6), and let p_2, p_3 be neighbors of v_j in B_{j+1} such that the paths $T_{v_{j+1}}^{j+1}[v_{j+1}, p_2]$ and $T_{u_{j+1}}^{j+1}[u_{j+1}, p_3]$ are disjoint (they exist by Lemma 3.6).
Set $T_1 \leftarrow T_1 + \{v_j, v_j p_1\}$, $T_2 \leftarrow T_2 + \{v_j, v_j p_2\}$, $T_3 \leftarrow T_3 + \{v_j, v_j p_3\}$, and $T_4 \leftarrow T_4 + \{v_j, v_j p_4\}$.

Case 3. $i = t$, or H_i is a down G_{i-1} -chain in G .

Let $H_i := v_0 B_1 v_1 \dots v_{k-1} B_k v_k$, with $v_0 = v_k = r$ when $i = t$, and $f(v_0) < f(v_k)$ when $i \neq t$.

For each 2-connected block B_j , let u_j, w_j denote the terminals of B_j^+ other than v_{j-1}, v_j , with $g(u_j) < g(w_j)$, and let $T_{v_{j-1}}^j, T_{v_j}^j, T_{u_j}^j, T_{w_j}^j$ denote the trees rooted, respectively, at v_{j-1}, v_j, u_j, w_j in the independent spanning $\{v_{j-1}, v_j, u_j, w_j\}$ -system of B_j^+ computed in Assumption 4.5.

Let $J := \{j : 1 \leq j \leq k, B_j \text{ is 2-connected}\}$, and let $\bar{J} := \{1, \dots, k\} - J$.

First, set

$$\begin{aligned} T_1 &\leftarrow T_1 \cup \left(\bigcup_{j \in J} T_{u_j}^j \right), \\ T_2 &\leftarrow T_2 \cup \left(\bigcup_{j \in J} T_{w_j}^j \right), \\ T_3 &\leftarrow T_3 \cup \left(\bigcup_{j \in \bar{J} - \{k\}} B_j \right) \cup \left(\bigcup_{j \in J} T_{v_{j-1}}^j \right), \text{ and} \\ T_4 &\leftarrow T_4 \cup \left(\bigcup_{j \in \bar{J} - \{1\}} B_j \right) \cup \left(\bigcup_{j \in J} T_{v_j}^j \right). \end{aligned}$$

Now for each $j = 1, \dots, k - 1$ add v_j and edges incident to v_j to T_1, T_2, T_3, T_4 according to the following cases (at this stage $v_0, v_k \notin V(T_1 \cup T_2)$).

Subcase 3.1. B_j and B_{j+1} are trivial blocks.

Let p_1, p_2 be neighbors of v_j in $V(G_{i-1})$ with $g(p_1)$ minimum (hence $g(p_1) < g(p_2)$).

Set $T_1 \leftarrow T_1 + \{v_j, v_j p_1\}$ and $T_2 \leftarrow T_2 + \{v_j, v_j p_2\}$.

Subcase 3.2. B_j is 2-connected and B_{j+1} is trivial.

- If v_j has no neighbor in $V(G_{i-1})$, then let p_1, p_2, p_3 be neighbors of v_j in B_j (they exist since G is 4-connected), and assume that p_1, p_3, p_2 is $(T_{u_j}^j, T_{w_j}^j)$ -ordered (this is possible by Lemma 3.4). By Lemma 3.7, $T_{u_j}^j[u_j, p_1]$, $T_{v_{j-1}}^j[v_{j-1}, p_3]$, and $T_{w_j}^j[w_j, p_2]$ are disjoint. If $k = 2$, then we also require that p_1, p_3, p_2 be the vertices q_1, q_2, q_3 , respectively, chosen in Subcase 3.2 of Algorithm Numbering g .

Set $T_1 \leftarrow T_1 + \{v_j, v_j p_1\}$, $T_2 \leftarrow T_2 + \{v_j, v_j p_2\}$, and $T_3 \leftarrow T_3 + \{v_j, v_j p_3\}$.

- If v_j has a neighbor in $V(G_{i-1})$, then let $x \in N_G(v_j) \cap V(G_{i-1})$ with $g(x)$ minimum.
 - If $g(x) > g(u_j)$, then let p_1, p_3 be neighbors of v_j in B_j such that the paths $T_{u_j}^j[u_j, p_1]$ and $T_{v_{j-1}}^j[v_{j-1}, p_3]$ are disjoint (they exist by Lemma 3.6). Set $T_1 \leftarrow T_1 + \{v_j, v_j p_1\}$, $T_2 \leftarrow T_2 + \{v_j, v_j x\}$, and $T_3 \leftarrow T_3 + \{v_j, v_j p_3\}$.
 - If $g(x) \leq g(u_j)$, then let p_2, p_3 be neighbors of v_j in B_j such that the paths $T_{w_j}^j[w_j, p_2]$ and $T_{v_{j-1}}^j[v_{j-1}, p_3]$ are disjoint (they exist by Lemma 3.6). Set $T_1 \leftarrow T_1 + \{v_j, v_j x\}$, $T_2 \leftarrow T_2 + \{v_j, v_j p_2\}$, and $T_3 \leftarrow T_3 + \{v_j, v_j p_3\}$.

Subcase 3.3. B_j is trivial and B_{j+1} is 2-connected.

- If v_j has no neighbor in $V(G_{i-1})$, then let p_1, p_2, p_4 be neighbors of v_j in B_{j+1} (they exist since G is 4-connected), and assume that p_1, p_4, p_2 is $(T_{u_{j+1}}^{j+1}, T_{w_{j+1}}^{j+1})$ -ordered (this is possible by Lemma 3.4). By Lemma 3.7, $T_{u_{j+1}}^{j+1}[u_{j+1}, p_1]$, $T_{v_{j+1}}^{j+1}[v_{j+1}, p_4]$, and $T_{w_{j+1}}^{j+1}[w_{j+1}, p_2]$ are disjoint. If $k = 2$, then

we also require that p_1, p_4, p_2 be the vertices q_1, q_2, q_3 , respectively, chosen in Subcase 3.2 of Algorithm Numbering g .

Set $T_1 \leftarrow T_1 + \{v_j, v_j p_1\}, T_2 \leftarrow T_2 + \{v_j, v_j p_2\}$, and $T_4 \leftarrow T_4 + \{v_j, v_j p_4\}$.

- If v_j has a neighbor in $V(G_{i-1})$, then let $x \in N_G(v_j) \cap V(G_{i-1})$ with $g(x)$ minimum.
 - If $g(x) > g(u_{j+1})$, then let p_1, p_4 be neighbors of v_j in B_{j+1} such that the paths $T_{u_{j+1}}^{j+1}[u_{j+1}, p_1]$ and $T_{v_{j+1}}^{j+1}[v_{j+1}, p_4]$ are disjoint (they exist by Lemma 3.6).
Set $T_1 \leftarrow T_1 + \{v_j, v_j p_1\}, T_2 \leftarrow T_2 + \{v_j, v_j x\}$, and $T_4 \leftarrow T_4 + \{v_j, v_j p_4\}$.
 - If $g(x) \leq g(u_{j+1})$, then let p_2, p_4 be neighbors of v_j in B_{j+1} such that the paths $T_{w_{j+1}}^{j+1}[w_{j+1}, p_2]$ and $T_{v_{j+1}}^{j+1}[v_{j+1}, p_4]$ are disjoint (they exist by Lemma 3.6).
Set $T_1 \leftarrow T_1 + \{v_j, v_j x\}, T_2 \leftarrow T_2 + \{v_j, v_j p_2\}$, and $T_4 \leftarrow T_4 + \{v_j, v_j p_4\}$.

Subcase 3.4. B_j and B_{j+1} are 2-connected.

Note that $g(u_j) < g(w_{j+1})$ or $g(u_{j+1}) < g(w_j)$.

- If $g(u_j) < g(w_{j+1})$, then let p_1, p_3 be neighbors of v_j in B_j such that the paths $T_{u_j}^j[u_j, p_1]$ and $T_{v_{j-1}}^j[v_{j-1}, p_3]$ are disjoint (they exist by Lemma 3.6), and let p_2, p_4 be neighbors of v_j in B_{j+1} such that the paths $T_{w_{j+1}}^{j+1}[w_{j+1}, p_2]$ and $T_{v_{j+1}}^{j+1}[v_{j+1}, p_4]$ are disjoint (they exist by Lemma 3.6).
Set $T_1 \leftarrow T_1 + \{v_j, v_j p_1\}, T_2 \leftarrow T_2 + \{v_j, v_j p_2\}, T_3 \leftarrow T_3 + \{v_j, v_j p_3\}$, and $T_4 \leftarrow T_4 + \{v_j, v_j p_4\}$.
- If $g(u_j) \geq g(w_{j+1})$, then $g(u_{j+1}) < g(w_j)$. Let p_2, p_3 be neighbors of v_j in B_j such that the paths $T_{w_j}^j[w_j, p_2]$ and $T_{v_{j-1}}^j[v_{j-1}, p_3]$ are disjoint (they exist by Lemma 3.6), and let p_1, p_4 be neighbors of v_j in B_{j+1} such that the paths $T_{u_{j+1}}^{j+1}[u_{j+1}, p_1]$ and $T_{v_{j+1}}^{j+1}[v_{j+1}, p_4]$ are disjoint (they exist by Lemma 3.6).
Set $T_1 \leftarrow T_1 + \{v_j, v_j p_1\}, T_2 \leftarrow T_2 + \{v_j, v_j p_2\}, T_3 \leftarrow T_3 + \{v_j, v_j p_3\}$, and $T_4 \leftarrow T_4 + \{v_j, v_j p_4\}$.

Case 4. H_i is a triangle G_{i-1} -chain in G .

Let $I(H_i) := \{v_1, v_2, v_3\}$, let $v_j x_j$ ($j = 1, 2, 3$) be the legs of H_i , and let $y_1, y_2, y_3 \in V(\bar{G}_i)$ such that $y_1 v_1, y_2 v_2, y_3 v_3 \in E(G)$. Assume that v_1, v_2, v_3 are labeled so that $g(x_1) < g(x_2) < g(x_3)$.

Update T_1, T_2, T_3, T_4 according to the following four possibilities.

- If $f(y_1) < f(y_2)$ and $f(y_1) < f(y_3)$ then set
 $T_1 \leftarrow T_1 + \{v_1, v_2, v_3, x_1 v_1, x_2 v_2, v_2 v_3\}, T_2 \leftarrow T_2 + \{v_1, v_2, v_3, x_3 v_3, v_3 v_1, v_3 v_2\},$
 $T_3 \leftarrow T_3 + \{v_1, v_2, v_3, y_1 v_1, v_1 v_2, v_1 v_3\}, T_4 \leftarrow T_4 + \{v_1, v_2, v_3, y_2 v_2, v_2 v_1, y_3 v_3\}.$
- If $f(y_2) < f(y_1)$ and $f(y_2) < f(y_3)$ then set
 $T_1 \leftarrow T_1 + \{v_1, v_2, v_3, x_1 v_1, x_2 v_2, v_1 v_3\}, T_2 \leftarrow T_2 + \{v_1, v_2, v_3, x_3 v_3, v_3 v_1, v_3 v_2\},$
 $T_3 \leftarrow T_3 + \{v_1, v_2, v_3, y_2 v_2, v_2 v_1, v_2 v_3\}, T_4 \leftarrow T_4 + \{v_1, v_2, v_3, y_1 v_1, v_1 v_2, y_3 v_3\}.$
- If $f(y_3) < f(y_1) < f(y_2)$ then set
 $T_1 \leftarrow T_1 + \{v_1, v_2, v_3, x_1 v_1, x_2 v_2, v_1 v_3\}, T_2 \leftarrow T_2 + \{v_1, v_2, v_3, x_3 v_3, v_3 v_1, v_3 v_2\},$
 $T_3 \leftarrow T_3 + \{v_1, v_2, v_3, y_1 v_1, v_1 v_2, y_3 v_3\}, T_4 \leftarrow T_4 + \{v_1, v_2, v_3, y_2 v_2, v_2 v_1, v_2 v_3\}.$
- If $f(y_3) < f(y_2) < f(y_1)$ then set
 $T_1 \leftarrow T_1 + \{v_1, v_2, v_3, x_1 v_1, x_2 v_2, v_2 v_3\}, T_2 \leftarrow T_2 + \{v_1, v_2, v_3, x_3 v_3, v_3 v_1, v_3 v_2\},$
 $T_3 \leftarrow T_3 + \{v_1, v_2, v_3, y_2 v_2, v_2 v_1, y_3 v_3\}, T_4 \leftarrow T_4 + \{v_1, v_2, v_3, y_1 v_1, v_1 v_2, v_1 v_3\}.$

6. Correctness of Algorithm Trees. In this section we will prove Theorem 1.1. More precisely, we will show that the subgraphs T_1, T_2, T_3, T_4 returned by Algorithm Trees are independent spanning trees of G rooted at r , and they can be computed in $O(|V(G)|^3)$ time.

Notation 6.1. Let G be a 4-connected graph, let $r \in V(G)$, and let $\mathcal{C} = (H_1, \dots, H_t)$ be a nonseparating chain decomposition of G rooted at r . Let $G_0 = \bar{G}_t = (\{r\}, \emptyset)$, and for $1 \leq i \leq t - 1$, let $G_i := G[\bigcup_{j=1}^i I(H_j)]$ and $\bar{G}_i = G - (V(G_i) - \{r\})$. Let T_1, T_2, T_3, T_4 denote the subgraphs returned by Algorithm Trees. Let D, D' denote the sets of vertices returned by Algorithm Numbering g and Algorithm Numbering f , respectively.

We start with a series of seven simple lemmas which follow from the cases of Algorithm Trees. The first lemma follows immediately by inspecting Case 1 of Algorithm Trees.

LEMMA 6.2. *Let $H_i := v_0B_1v_1B_2v_2$ be an elementary G_{i-1} -chain in G , with $g(v_0) < g(v_2)$. Then v_1 has neighbors v'_0, v'_2 in $V(\bar{G}_i)$, with $f(v'_0) < f(v'_2)$, such that*

- (1) $E(T_1 \cap H_i) = \{v_0v_1\}$ and $E(T_2 \cap H_i) = \{v_1v_2\}$, and
- (2) $E(T_3 \cap H'_i) = \{v'_0v_1\}$ and $E(T_4 \cap H'_i) = \{v_1v'_2\}$, where $H'_i = v'_0B'_1v_1B'_2v'_2$ is an elementary \bar{G}_i -chain in G .

The next lemma follows by inspecting Case 2 (for $i = 1$) of Algorithm Trees.

LEMMA 6.3. *Let $H_1 := v_0B_1v_1 \dots v_{k-1}B_kv_k$, with $v_0 = v_k = r$, and for each 2-connected B_j , let u_j, w_j denote the terminals of B_j^+ other than v_{j-1}, v_j , with $f(u_j) < f(w_j)$. Let H_1^+ be the graph obtained from H_1 by adding $N_G(H_1 - r) - \{r\}$ and the edges of G from $V(H_1)$ to $N_G(H_1 - r) - \{r\}$. Then*

- (1) $T_1 \cap H_1$ is a spanning tree of H_1 rooted at r and contains no edge from r to $N_{B_k}(r)$,
- (2) $T_2 \cap H_1$ is a spanning tree of H_1 rooted at r and contains no edge from r to $N_{B_1}(r)$,
- (3) $T_3 \cap (H_1^+ - r)$ is a spanning forest of $H_1^+ - r$, and each component of $T_3 \cap (H_1^+ - r)$ either is a tree in $B_j^+ - w_j$ rooted at u_j for some $j \in \{1, \dots, k\}$ or is induced by a single edge with one end in $V(\bar{G}_1)$ and the other in $\{v_1, \dots, v_{k-1}\}$, and
- (4) $T_4 \cap (H_1^+ - r)$ is a spanning forest of $H_1^+ - r$, and each component of $T_4 \cap (H_1^+ - r)$ either is a tree in $B_j^+ - u_j$ rooted at w_j for some $j \in \{1, \dots, k\}$ or is induced by a single edge with one end in $V(\bar{G}_1)$ and the other in $\{v_1, \dots, v_{k-1}\}$.

By inspecting Case 2 (for $i \neq 1$) of Algorithms Trees, we have the following lemma.

LEMMA 6.4. *Let $H_i := v_0B_1v_1 \dots v_{k-1}B_kv_k$ be an up G_{i-1} -chain in G ($2 \leq i \leq t - 1$), with $g(v_0) < g(v_k)$, and for each 2-connected block B_j , let u_j, w_j denote the terminals of B_j^+ other than v_{j-1}, v_j , with $f(u_j) < f(w_j)$. Let H_i^+ be the graph obtained from H_i by adding $N_G(H_i - \{v_0, v_k\}) - \{v_0, v_k\}$ and the edges of G from $V(H_i)$ to $N_G(H_i - \{v_0, v_k\}) - \{v_0, v_k\}$. Then*

- (1) $T_1 \cap (H_i - v_k)$ is a spanning tree of $H_i - v_k$ rooted at v_0 , and T_1 contains no edge from v_k to $N_{B_k}(v_k)$,
- (2) $T_2 \cap (H_i - v_0)$ is a spanning tree of $H_i - v_0$ rooted at v_k , and T_2 contains no edge from v_0 to $N_{B_1}(v_0)$,
- (3) $T_3 \cap (H_i^+ - \{v_0, v_k\})$ is a spanning forest of $H_i^+ - \{v_0, v_k\}$, and each component of $T_3 \cap (H_i^+ - \{v_0, v_k\})$ either is a tree in $B_j^+ - w_j$ rooted at u_j for some $j \in \{1, \dots, k\}$ or is induced by a single edge with one end in $V(\bar{G}_i)$ and the other in $\{v_1, \dots, v_{k-1}\}$, and
- (4) $T_4 \cap (H_i^+ - \{v_0, v_k\})$ is a spanning forest of $H_i^+ - \{v_0, v_k\}$, and each component of $T_4 \cap (H_i^+ - \{v_0, v_k\})$ either is a tree in $B_j^+ - u_j$ rooted at w_j for some $j \in \{1, \dots, k\}$ or is induced by a single edge with one end in $V(\bar{G}_i)$ and the other in $\{v_1, \dots, v_{k-1}\}$.

By a simple inspection of Case 3 (for $i = t$) of Algorithm Trees, we have the following lemma.

LEMMA 6.5. *Let $H_t := v_0 B_1 v_1 \dots v_{k-1} B_k v_k$, with $v_0 = v_k = r$, and for each 2-connected B_j , let u_j, w_j denote the terminals of B_j^+ other than v_{j-1}, v_j , with $g(u_j) < g(w_j)$. Let H_t^+ be the graph obtained from H_t by adding $N_G(H_t - r) - \{r\}$ and the edges of G from $V(H_t)$ to $N_G(H_t - r) - \{r\}$. Then*

- (1) $T_1 \cap (H_t^+ - r)$ is a spanning forest of $H_t^+ - r$, and each component of $T_1 \cap (H_t^+ - r)$ either is a tree in $B_j^+ - w_j$ rooted at u_j for some $j \in \{1, \dots, k\}$ or is induced by a single edge with one end in $V(G_{t-1})$ and the other in $\{v_1, \dots, v_{k-1}\}$,
- (2) $T_2 \cap (H_t^+ - r)$ is a spanning forest of $H_t^+ - r$, and each component of $T_2 \cap (H_t^+ - r)$ either is a tree in $B_j^+ - u_j$ rooted at w_j for some $j \in \{1, \dots, k\}$ or is induced by a single edge with one end in $V(G_{t-1})$ and the other in $\{v_1, \dots, v_{k-1}\}$,
- (3) $T_3 \cap H_t$ is a spanning tree of H_t rooted at r and contains no edge from r to $N_{B_k}(r)$, and
- (4) $T_4 \cap H_t$ is a spanning tree of H_t rooted at r and contains no edge from r to $N_{B_1}(r)$.

The next lemma follows from a simple inspection of Case 3 (for $i \neq t$) of Algorithm Trees.

LEMMA 6.6. *Let $H_i := v_0 B_1 v_1 \dots v_{k-1} B_k v_k$ be a down G_{i-1} -chain in G ($2 \leq i \leq t-1$), with $f(v_0) < f(v_k)$, and for each 2-connected block B_j , let u_j, w_j denote the terminals of B_j^+ other than v_{j-1}, v_j , with $g(u_j) < g(w_j)$. Let H_i^+ be the graph obtained from H_i by adding $N_G(H_i - \{v_0, v_k\}) - \{v_0, v_k\}$ and the edges of G from $V(H_i)$ to $N_G(H_i - \{v_0, v_k\}) - \{v_0, v_k\}$. Then*

- (1) $T_1 \cap (H_i^+ - \{v_0, v_k\})$ is a spanning forest of $H_i^+ - \{v_0, v_k\}$, and each component of $T_1 \cap (H_i^+ - \{v_0, v_k\})$ either is a tree in $B_j^+ - w_j$ rooted at u_j for some $j \in \{1, \dots, k\}$ or is induced by a single edge with one end in $V(G_{i-1})$ and the other in $\{v_1, \dots, v_{k-1}\}$,
- (2) $T_2 \cap (H_i^+ - \{v_0, v_k\})$ is a spanning forest of $H_i^+ - \{v_0, v_k\}$, and each component of $T_2 \cap (H_i^+ - \{v_0, v_k\})$ either is a tree in $B_j^+ - u_j$ rooted at w_j for some $j \in \{1, \dots, k\}$ or is induced by a single edge with one end in $V(G_{i-1})$ and the other in $\{v_1, \dots, v_{k-1}\}$,
- (3) $T_3 \cap (H_i - v_k)$ is a spanning tree of $H_i - v_k$ rooted at v_0 , and T_3 contains no edge from v_k to $N_{B_k}(v_k)$, and
- (4) $T_4 \cap (H_i - v_0)$ is a spanning tree of $H_i - v_0$ rooted at v_k , and T_4 contains no edge from v_0 to $N_{B_1}(v_0)$.

Finally, by a simple inspection of Case 4 of Algorithm Trees, we have the following lemma.

LEMMA 6.7. *Let H_i be a triangle G_{i-1} -chain in G ($2 \leq i \leq t-1$). Let $I(H_i) := \{v_1, v_2, v_3\}$, let $y_1, y_2, y_3 \in V(\bar{G}_i)$ such that $y_1 v_1, y_2 v_2, y_3 v_3 \in E(G)$, and let $v_j x_j$ ($j = 1, 2, 3$) be the legs of H_i , with $g(x_1) < g(x_2) < g(x_3)$. Let $H_i^+ := H_i + \{y_1, y_2, y_3, y_1 v_1, y_2 v_2, y_3 v_3\}$.*

- *If $f(y_1) < f(y_2)$ and $f(y_1) < f(y_3)$, then*
 $E(T_1 \cap H_i^+) = \{x_1 v_1, x_2 v_2, v_2 v_3\}$, $E(T_2 \cap H_i^+) = \{x_3 v_3, v_3 v_1, v_3 v_2\}$,
 $E(T_3 \cap H_i^+) = \{y_1 v_1, v_1 v_2, v_1 v_3\}$, and $E(T_4 \cap H_i^+) = \{y_2 v_2, v_2 v_1, y_3 v_3\}$.
- *If $f(y_2) < f(y_1)$ and $f(y_2) < f(y_3)$, then*
 $E(T_1 \cap H_i^+) = \{x_1 v_1, x_2 v_2, v_1 v_3\}$, $E(T_2 \cap H_i^+) = \{x_3 v_3, v_3 v_1, v_3 v_2\}$,
 $E(T_3 \cap H_i^+) = \{y_2 v_2, v_2 v_1, v_2 v_3\}$, and $E(T_4 \cap H_i^+) = \{y_1 v_1, v_1 v_2, y_3 v_3\}$.

- If $f(y_3) < f(y_1) < f(y_2)$, then
 $E(T_1 \cap H_i^+) = \{x_1v_1, x_2v_2, v_1v_3\}$, $E(T_2 \cap H_i^+) = \{x_3v_3, v_3v_1, v_3v_2\}$,
 $E(T_3 \cap H_i^+) = \{y_1v_1, v_1v_2, y_3v_3\}$, and $E(T_4 \cap H_i^+) = \{y_2v_2, v_2v_1, v_2v_3\}$.
- If $f(y_3) < f(y_2) < f(y_1)$, then
 $E(T_1 \cap H_i^+) = \{x_1v_1, x_2v_2, v_2v_3\}$, $E(T_2 \cap H_i^+) = \{x_3v_3, v_3v_1, v_3v_2\}$,
 $E(T_3 \cap H_i^+) = \{y_2v_2, v_2v_1, y_3v_3\}$, and $E(T_4 \cap H_i^+) = \{y_1v_1, v_1v_2, v_1v_3\}$.

We can now show that T_1, T_2, T_3 , and T_4 are spanning trees of G .

LEMMA 6.8. For every $i = 1, \dots, t$, $T_1 \cap G_i$ and $T_2 \cap G_i$ are spanning trees of G_i .

Proof. Note that every $v \in V(G) - \{r\}$ is an internal vertex of some chain H_i in \mathcal{C} . The result follows by induction on i with the help of (1) of Lemma 6.2, (1) and (2) of Lemma 6.3, (1) and (2) of Lemma 6.4, (1) and (2) of Lemma 6.5, (1) and (2) of Lemma 6.6, and Lemma 6.7. \square

LEMMA 6.9. For every $i = t, \dots, 1$, $T_3 \cap \bar{G}_i$ and $T_4 \cap \bar{G}_i$ are spanning trees of \bar{G}_i .

Proof. The result follows by induction on $t - i$ with the help of (2) of Lemma 6.2, (3) and (4) of Lemma 6.3, (3) and (4) of Lemma 6.4, (3) and (4) of Lemma 6.5, (3) and (4) of Lemma 6.6, and Lemma 6.7. \square

Lemmas 6.8 and 6.9 imply the following.

COROLLARY 6.10. T_1, T_2, T_3, T_4 are spanning trees of G .

Now we proceed to show that T_1, T_2, T_3, T_4 are independent spanning trees of G rooted at r . The proof consists of several lemmas.

LEMMA 6.11. For any $1 \leq i \leq t$ and for any $v \in I(H_i) - \{r\}$, there exist vertices z_1, z_2, z_3, z_4 such that

- (1) $z_1, z_2 \in V(G_{i-1})$, and either $z_1 = z_2 = r$ or $g(z_1) < g(z_2)$ (and $g(z_1) < g(v) < g(z_2)$ if $v \in D$),
- (2) $z_3, z_4 \in V(\bar{G}_i)$, and either $z_3 = z_4 = r$ or $f(z_3) < f(z_4)$ (and $f(z_3) < f(v) < f(z_4)$ if $v \in D'$), and
- (3) $T_i[z_i, v]$, $i = 1, 2, 3, 4$, are internally disjoint paths in G , and $V(T_i[z_i, v]) - z_i \subseteq I(H_i)$.

Proof. Let $1 \leq i \leq t$ and $v \in I(H_i) - \{r\}$. We consider the four cases of Algorithm Trees.

Case 1. H_i is an elementary G_{i-1} -chain in G .

In this case, $2 \leq i \leq t - 1$. Let $H_i := v_0B_1v_1B_2v_2$, with $g(v_0) < g(v_2)$. This is the same as in Case 1 of Algorithm Trees. Then $v_0, v_2 \in V(G_{i-1})$, $v = v_1$, and by Case 1 of Algorithm Numbering g , we have $g(v_0) < g(v_1) < g(v_2)$. By Lemma 6.2, there exist $v'_0, v'_2 \in V(\bar{G}_i)$, with $f(v'_0) < f(v'_2)$, such that $v_0v \in E(T_1)$, $v_2v \in E(T_2)$, $v'_0v \in E(T_3)$, and $v'_2v \in E(T_4)$. By Case 1 of Algorithm Numbering f , $f(v'_0) < f(v_1) < f(v'_2)$. Thus, the result follows by taking $z_1 := v_0$, $z_2 := v_2$, $z_3 := v'_0$, and $z_4 := v'_2$.

Case 2. $i = 1$, or H_i is an up G_{i-1} -chain in G but not an elementary G_{i-1} -chain.

Let $H_i := v_0B_1v_1 \dots v_{k-1}B_kv_k$, with $v_0 = v_k = r$ when $i = 1$, and $g(v_0) < g(v_k)$ when $i \neq 1$. For each 2-connected B_j , let u_j, w_j denote the terminals of B_j^+ other than v_{j-1}, v_j , with $f(u_j) < f(w_j)$, and let $T_{v_{j-1}}^j, T_{v_j}^j, T_{u_j}^j, T_{w_j}^j$ denote the trees rooted, respectively, at v_{j-1}, v_j, u_j, w_j in the independent spanning $\{v_{j-1}, v_j, u_j, w_j\}$ -system of B_j^+ computed in Assumption 4.5. This is the same as in Case 2 of Algorithm Trees.

Let $j \in \{1, \dots, k - 1\}$. If $i = 1$, then by (1) and (2) of Lemma 6.3, $T_1[r, v_j] \subseteq \bigcup_{l=1}^j B_l$ and $T_2[r, v_j] \subseteq \bigcup_{l=j+1}^k B_l$. If $i \neq 1$, then v_j is a cut vertex of H_i , and hence, by (1) and (2) of Lemma 6.4, $T_1[v_0, v_j] \subseteq \bigcup_{l=1}^j B_l$ and $T_2[v_k, v_j] \subseteq \bigcup_{l=j+1}^k B_l$.

First, let us consider the case when $v \neq v_j$ for $j = 1, \dots, k - 1$. Thus, there exists some j , $1 \leq j \leq k$, such that B_j is 2-connected and $v \in V(B_j) - \{v_{j-1}, v_j\}$. By Case 2 of Algorithm Numbering g , we know that $g(v_0) \leq g(v_{j-1}) < g(v_j) \leq$

$g(v_k)$, and if $v \in D$, then $g(v_0) \leq g(v_{j-1}) < g(v) < g(v_j) \leq g(v_k)$. Furthermore, $T_{v_{j-1}}^j[v_{j-1}, v]$, $T_{v_j}^j[v_j, v]$, $T_{u_j}^j[u_j, v]$, and $T_{w_j}^j[w_j, v]$ are internally disjoint, because $\{T_{v_{j-1}}^j, T_{v_j}^j, T_{u_j}^j, T_{w_j}^j\}$ is an independent spanning $\{v_{j-1}, v_j, u_j, w_j\}$ -system of B_j^+ . By the construction in Case 2 of Algorithm Trees, $T_1[v_{j-1}, v] = T_{v_{j-1}}^j[v_{j-1}, v]$, $T_2[v_j, v] = T_{v_j}^j[v_j, v]$, $T_3[u_j, v] = T_{u_j}^j[u_j, v]$, and $T_4[w_j, v] = T_{w_j}^j[w_j, v]$. By Case 3 of Algorithm Numbering f , if $v \in D'$, then $f(u_j) < f(v) < f(w_j)$. Moreover, $T_1[v_0, v_{j-1}] \subseteq \bigcup_{l=1}^{j-1} B_l$ and $T_2[v_k, v_j] \subseteq \bigcup_{l=j+1}^k B_l$. Let $z_1 := v_0, z_2 := v_k, z_3 := u_j$, and $z_4 := w_j$. Clearly, (1)–(3) hold.

So assume that $v = v_j$ for some $j, 1 \leq j \leq k-1$. Let $z_1 := v_0$ and $z_2 := v_k$. Then by Case 2 of Algorithm Numbering g , $g(z_1) < g(v) < g(z_2)$. We will define z_3 and z_4 and prove that (1)–(3) hold. We do this by analyzing how Algorithm Trees chooses the neighbors p_3, p_4 of v_j in the trees T_3, T_4 , respectively.

Subcase 2.1. B_j and B_{j+1} are trivial (Subcase 2.1 in Algorithm Trees).

Then Algorithm Trees chooses neighbors p_3, p_4 of v_j in $V(\bar{G}_i)$ with $f(p_3)$ minimum (and hence $f(p_3) < f(p_4)$). If $v_j \in D'$, then by Case 3 of Algorithm Numbering f , we have $f(p_3) < f(v_j) < f(p_4)$. Let $z_3 := p_3$ and $z_4 := p_4$. Clearly, (1)–(3) hold.

Subcase 2.2. B_j is 2-connected and B_{j+1} is trivial (Subcase 2.2 in Algorithm Trees).

- If v_j has no neighbor in $V(\bar{G}_i)$, then Algorithm Trees chooses three neighbors p_1, p_3, p_4 of v_j in B_j such that $T_{v_{j-1}}^j[v_{j-1}, p_1]$, $T_{u_j}^j[u_j, p_3]$, and $T_{w_j}^j[w_j, p_4]$ are disjoint. By construction, $T_1[v_{j-1}, v_j] = T_{v_{j-1}}^j[v_{j-1}, p_1] + \{v_j, v_j p_1\}$, $T_3[u_j, v_j] = T_{u_j}^j[u_j, p_3] + \{v_j, v_j p_3\}$, and $T_4[w_j, v_j] = T_{w_j}^j[w_j, p_4] + \{v_j, v_j p_4\}$. Moreover, $T_1[v_0, v_{j-1}] \subseteq \bigcup_{l=1}^{j-1} B_l$ and $T_2[v_k, v_j] \subseteq \bigcup_{l=j+1}^k B_l$. Therefore, $T_1[v_0, v_j]$, $T_2[v_k, v_j]$, $T_3[u_j, v_j]$, and $T_4[w_j, v_j]$ are internally disjoint. If $v_j \in D'$, then by Case 3 of Algorithm Numbering f , we have $f(u_j) < f(v) < f(w_j)$. Let $z_3 := u_j$ and $z_4 := w_j$. Clearly, (1)–(3) hold.
- If v_j has a neighbor in $V(\bar{G}_i)$, then Algorithm Trees chooses a vertex $x \in N_G(v_j) \cap V(\bar{G}_i)$ with $f(x)$ minimum.
 - If $f(x) > f(u_j)$, then the algorithm chooses neighbors p_1, p_3 of v_j in B_j such that $T_{v_{j-1}}^j[v_{j-1}, p_1]$ and $T_{u_j}^j[u_j, p_3]$ are disjoint. By construction, $T_1[v_{j-1}, v_j] = T_{v_{j-1}}^j[v_{j-1}, p_1] + \{v_j, v_j p_1\}$, $T_3[u_j, v_j] = T_{u_j}^j[u_j, p_3] + \{v_j, v_j p_3\}$, and $T_4[x, v_j]$ is induced by the edge xv_j . Moreover, $T_1[v_0, v_{j-1}] \subseteq \bigcup_{l=1}^{j-1} B_l$ and $T_2[v_k, v_j] \subseteq \bigcup_{l=j+1}^k B_l$. Therefore, $T_1[v_0, v_j]$, $T_2[v_k, v_j]$, $T_3[u_j, v_j]$, and $T_4[x, v_j]$ are internally disjoint. If $v_j \in D'$, then by Case 3 of Algorithm Numbering f , we have $f(u_j) < f(v) < f(x)$. Let $z_3 := u_j$ and $z_4 := x$. Clearly, (1)–(3) hold.
 - If $f(x) \leq f(u_j)$, then Algorithm Trees chooses neighbors p_1, p_4 of v_j in B_j such that $T_{v_{j-1}}^j[v_{j-1}, p_1]$ and $T_{w_j}^j[w_j, p_4]$ are disjoint. By construction, $T_1[v_{j-1}, v_j] = T_{v_{j-1}}^j[v_{j-1}, p_1] + \{v_j, v_j p_1\}$, $T_4[w_j, v_j] = T_{w_j}^j[w_j, p_4] + \{v_j, v_j p_4\}$, and $T_3[x, v_j]$ is induced by the edge xv_j . Moreover, $T_1[v_0, v_{j-1}] \subseteq \bigcup_{l=1}^{j-1} B_l$ and $T_2[v_k, v_j] \subseteq \bigcup_{l=j+1}^k B_l$. Therefore, $T_1[v_0, v_j]$, $T_2[v_k, v_j]$, $T_3[x, v_j]$, and $T_4[w_j, v_j]$ are internally disjoint. If $v_j \in D'$, then by Case 3 of Algorithm Numbering f , we have $f(x) < f(v) < f(w_j)$. Let $z_3 := x$ and $z_4 := w_j$. Clearly, (1)–(3) hold.

Subcase 2.3. B_j is trivial and B_{j+1} is 2-connected (Subcase 2.3 in Algorithm Trees).

In this case, if $v_j \in D'$, then $j = 1$ by Case 3 of Algorithm Numbering f . The arguments for the proof are similar to Subcase 2.2, and we indicate only the choice of

z_3 and z_4 . In each case below, one can show that (1)–(3) hold for the corresponding choice of z_3, z_4 .

- If v_j has no neighbor in $V(\bar{G}_i)$, then let $z_3 := u_{j+1}$ and $z_4 := w_{j+1}$.
- If v_j has a neighbor in $V(\bar{G}_i)$, then Algorithm Trees chooses a vertex $x \in N_G(v_j) \cap V(\bar{G}_i)$ with $f(x)$ minimum.
 - If $f(x) > f(u_{j+1})$, then let $z_3 := u_{j+1}$ and $z_4 := x$.
 - If $f(x) \leq f(u_{j+1})$, then let $z_3 := x$ and $z_4 := w_{j+1}$.

Subcase 2.4. Both B_j and B_{j+1} are 2-connected (Subcase 2.4 in Algorithm Trees).

Since G is 4-connected and $(B_j^+, v_{j-1}, u_j, v_j, w_j)$ and $(B_{j+1}^+, v_j, u_{j+1}, v_{j+1}, w_{j+1})$ are both planar, $v_j \notin N_{B_j}(v_{j-1}) \cup N_{B_{j+1}}(v_{j+1})$. Hence, $v_j \notin D'$ by Case 3 of Algorithm Numbering f . Note that $f(u_j) < f(w_{j+1})$ or $f(u_{j+1}) < f(w_j)$.

- If $f(u_j) < f(w_{j+1})$, then Algorithm Trees chooses neighbors p_1, p_3 of v_j in B_j such that $T_{v_{j-1}}^j[v_{j-1}, p_1], T_{u_j}^j[u_j, p_3]$ are disjoint and neighbors p_2, p_4 of v_j in B_{j+1} such that $T_{v_{j+1}}^j[v_{j+1}, p_2], T_{w_{j+1}}^j[w_{j+1}, p_4]$ are disjoint. By construction, $T_1[v_{j-1}, v_j] = T_{v_{j-1}}^j[v_{j-1}, p_1] + \{v_j, v_j p_1\}$, $T_3[u_j, v_j] = T_{u_j}^j[u_j, p_3] + \{v_j, v_j p_3\}$, $T_2[v_{j+1}, v_j] = T_{v_{j+1}}^{j+1}[v_{j+1}, p_2] + \{v_j, v_j p_2\}$, and $T_4[w_{j+1}, v_j] = T_{w_{j+1}}^{j+1}[w_{j+1}, p_4] + \{v_j, v_j p_4\}$. Moreover, $T_1[v_0, v_{j-1}] \subseteq \bigcup_{l=1}^{j-1} B_l$ and $T_2[v_k, v_{j+1}] \subseteq \bigcup_{l=j+2}^k B_l$. Thus, $T_1[v_0, v_j], T_2[v_k, v_j], T_3[u_j, v_j]$, and $T_4[w_{j+1}, v_j]$ are internally disjoint. Let $z_3 := u_j$ and $z_4 := w_{j+1}$. Clearly, (1)–(3) hold.
- If $f(u_j) \geq f(w_{j+1})$, then $f(u_{j+1}) < f(w_j)$, and Algorithm Trees chooses neighbors p_1, p_4 of v_j in B_j such that $T_{v_{j-1}}^j[v_{j-1}, p_1]$ and $T_{w_j}^j[w_j, p_4]$ are disjoint and neighbors p_2, p_3 of v_j in B_{j+1} such that $T_{v_{j+1}}^{j+1}[v_{j+1}, p_2]$ and $T_{u_{j+1}}^{j+1}[u_{j+1}, p_3]$ are disjoint. Let $z_3 := u_{j+1}$ and $z_4 := w_j$. One can show as in the above paragraph that $T_1[v_0, v_j], T_2[v_k, v_j], T_3[u_{j+1}, v_j]$, and $T_4[w_j, v_j]$ are internally disjoint and (1)–(3) hold.

Case 3. $i = t$, or H_i is a down G_{i-1} -chain in G but not an elementary G_{i-1} -chain.

Let $H_i := v_0 B_1 v_1 \dots v_{k-1} B_k v_k$, with $v_0 = v_k = r$ when $i = t$, and $f(v_0) < f(v_k)$ when $i \neq t$. For each 2-connected B_j , let u_j, w_j denote the terminals of B_j^+ other than v_{j-1}, v_j , with $g(u_j) < g(w_j)$, and let $T_{v_{j-1}}^j, T_{v_j}^j, T_{u_j}^j, T_{w_j}^j$ denote the trees rooted, respectively, at v_{j-1}, v_j, u_j, w_j in the independent spanning $\{v_{j-1}, v_j, u_j, w_j\}$ -system of B_j^+ computed in Assumption 4.5. This is the same as in Case 3 of Algorithm Trees.

Let $j \in \{1, \dots, k-1\}$. If $i = t$, then by (3) and (4) of Lemma 6.5, $T_3[v_0, v_j] \subseteq \bigcup_{l=1}^j B_l$ and $T_4[v_k, v_j] \subseteq \bigcup_{l=j+1}^k B_l$. If $i \neq t$, then v_j is a cut vertex of H_i , and hence, by (3) and (4) of Lemma 6.6, $T_3[v_0, v_j] \subseteq \bigcup_{l=1}^j B_l$ and $T_4[v_k, v_j] \subseteq \bigcup_{l=j+1}^k B_l$.

First, let us consider the case when $v \neq v_j$ for $j = 1, \dots, k-1$. Thus, there exists some j , $1 \leq j \leq k$, such that B_j is 2-connected and $v \in V(B_j) - \{v_{j-1}, v_j\}$. By Case 2 of Algorithm Numbering f , we know that $f(v_0) \leq f(v_{j-1}) < f(v_j) \leq f(v_k)$, and if $v_j \in D'$, then $f(v_0) \leq f(v_{j-1}) < f(v) < f(v_j) \leq f(v_k)$. Furthermore, $T_{v_{j-1}}^j[v_{j-1}, v], T_{v_j}^j[v_j, v], T_{u_j}^j[u_j, v]$, and $T_{w_j}^j[w_j, v]$ are internally disjoint because $\{T_{v_{j-1}}^j, T_{v_j}^j, T_{u_j}^j, T_{w_j}^j\}$ is an independent spanning $\{v_{j-1}, v_j, u_j, w_j\}$ -system of B_j^+ . By the construction in Case 3 of Algorithm Trees, $T_1[u_j, v] = T_{u_j}^j[u_j, v]$, $T_2[w_j, v] = T_{w_j}^j[w_j, v]$, $T_3[v_{j-1}, v] = T_{v_{j-1}}^j[v_{j-1}, v]$, and $T_4[v_j, v] = T_{v_j}^j[v_j, v]$. Moreover, $T_3[v_0, v_{j-1}] \subseteq \bigcup_{l=1}^{j-1} B_l$ and $T_4[v_k, v_j] \subseteq \bigcup_{l=j+1}^k B_l$. Let $z_1 := u_j, z_2 := w_j, z_3 := v_0$, and $z_4 := v_k$. Clearly, (1)–(3) hold.

So assume that $v = v_j$ for some j , $1 \leq j \leq k-1$. Let $z_3 := v_0$ and $z_4 := v_k$. By Case 2 of Algorithm Numbering f , we have $f(z_2) < f(v) < f(z_4)$. We will define z_1 and z_2 and prove that (1)–(3) hold. We do this by analyzing how Algorithm Trees chooses the neighbors p_1, p_2 of v_j in the trees T_1, T_2 , respectively.

Subcase 3.1. B_j and B_{j+1} are trivial (Subcase 3.1 in Algorithm Trees).

Then Algorithm Trees chooses neighbors p_1, p_2 of v_j in $V(G_{i-1})$ with $g(p_1)$ minimum (and so $g(p_1) < g(p_2)$). By Subcase 3.3 of Algorithm Numbering g , we have $g(p_1) < g(v) < g(p_2)$. Let $z_1 := p_1$ and $z_2 := p_2$. Clearly, (1)–(3) hold.

Subcase 3.2. B_j is 2-connected and B_{j+1} is trivial (Subcase 3.2 in Algorithm Trees).

- If v_j has no neighbor in $V(G_{i-1})$, then Algorithm Trees chooses three neighbors p_1, p_2, p_3 of v_j in B_j such that $T_{u_j}^j[u_j, p_1]$, $T_{w_j}^j[w_j, p_2]$, and $T_{v_{j-1}}^j[v_{j-1}, p_3]$ are disjoint. By construction, $T_1[u_j, v_j] = T_{u_j}^j[u_j, p_1] + \{v_j, v_j p_1\}$, $T_2[w_j, v_j] = T_{w_j}^j[w_j, p_2] + \{v_j, v_j p_2\}$, and $T_3[v_{j-1}, v_j] = T_{v_{j-1}}^j[v_{j-1}, p_3] + \{v_j, v_j p_3\}$. Moreover, $T_3[v_0, v_{j-1}] \subseteq \bigcup_{l=1}^{j-1} B_l$ and $T_4[v_k, v_j] \subseteq \bigcup_{l=j+1}^k B_l$. Therefore, $T_1[u_j, v_j]$, $T_2[w_j, v_j]$, $T_3[v_0, v_j]$, and $T_4[v_k, v_j]$ are internally disjoint. In this case, if $v_j \in D$, then by Case 3 of Algorithm Numbering g , we have $j = k - 1$ and $g(u_j) < g(v) < g(w_j)$. Let $z_1 := u_j$ and $z_2 := w_j$. Clearly, (1)–(3) hold.
- If v_j has a neighbor in $V(G_{i-1})$, then Algorithm Trees chooses a vertex $x \in N_G(v_j) \cap V(G_{i-1})$ with $g(x)$ minimum.
 - If $g(x) > g(u_j)$, then the algorithm chooses neighbors p_1, p_3 of v_j in B_j such that $T_{u_j}^j[u_j, p_1]$ and $T_{v_{j-1}}^j[v_{j-1}, p_3]$ are disjoint. By construction, $T_1[u_j, v_j] = T_{u_j}^j[u_j, p_1] + \{v_j, v_j p_1\}$, $T_3[v_{j-1}, v_j] = T_{v_{j-1}}^j[v_{j-1}, p_3] + \{v_j, v_j p_3\}$, and $T_2[x, v_j]$ is induced by the edge xv_j . Moreover, $T_3[v_0, v_{j-1}] \subseteq \bigcup_{l=1}^{j-1} B_l$ and $T_4[v_k, v_j] \subseteq \bigcup_{l=j+1}^k B_l$. Therefore, $T_1[u_j, v_j]$, $T_2[x, v_j]$, $T_3[v_0, v_j]$, and $T_4[v_k, v_j]$ are internally disjoint. If $v_j \in D$, then by Case 3 of Algorithm Numbering g , we have $j = k - 1$ and $g(u_j) < g(v) < g(x)$. Let $z_1 := u_j$ and $z_4 := x$. Clearly, (1)–(3) hold.
 - If $g(x) \leq g(u_j)$, then Algorithm Trees chooses neighbors p_2, p_3 of v_j in B_j such that $T_{w_j}^j[w_j, p_2]$ and $T_{v_{j-1}}^j[v_{j-1}, p_3]$ are disjoint. By construction, $T_2[w_j, v_j] = T_{w_j}^j[w_j, p_2] + \{v_j, v_j p_2\}$, $T_3[v_{j-1}, v_j] = T_{v_{j-1}}^j[v_{j-1}, p_3] + \{v_j, v_j p_3\}$, and $T_1[x, v_j]$ is induced by the edge xv_j . Moreover, $T_3[v_0, v_{j-1}] \subseteq \bigcup_{l=1}^{j-1} B_l$ and $T_4[v_k, v_j] \subseteq \bigcup_{l=j+1}^k B_l$. Therefore, $T_1[x, v_j]$, $T_2[w_j, v_j]$, $T_3[v_0, v_j]$, and $T_4[v_k, v_j]$, are internally disjoint. If $v_j \in D$, then by Case 3 of Algorithm Numbering g , we have $j = k - 1$ and $g(x) < g(v) < g(w_j)$. Let $z_1 := x$ and $z_2 := w_j$. Clearly, (1)–(3) hold.

Subcase 3.3. B_j is trivial and B_{j+1} is 2-connected (Subcase 3.3 in Algorithm Trees).

In this case, if $v_j \in D$, then $j = 1$ by Case 3 of Algorithm Numbering g . The arguments for this case are similar to Subcase 3.2, and we indicate only the choice of z_1 and z_2 . In each case below, one can show that (1)–(3) hold for the corresponding choice of z_1, z_2 .

- If v_j has no neighbor in $V(G_{i-1})$, then let $z_1 := u_{j+1}$ and $z_2 := w_{j+1}$.
- If v_j has a neighbor in $V(G_{i-1})$, then Algorithm Trees chooses a vertex $x \in N_G(v_j) \cap V(G_{i-1})$ with $g(x)$ minimum.
 - If $g(x) > g(u_{j+1})$, then let $z_1 := u_{j+1}$ and $z_2 := x$.
 - If $g(x) \leq g(u_{j+1})$, then let $z_1 := x$ and $z_2 := w_{j+1}$.

Subcase 3.4. B_j and B_{j+1} are 2-connected (Subcase 3.4 in Algorithm Trees).

Since G is 4-connected and $(B_j^+, v_{j-1}, u_j, v_j, w_j)$ and $(B_{j+1}^+, v_j, u_{j+1}, v_{j+1}, w_{j+1})$ are both planar, $v_j \notin N_{B_j}(v_{j-1}) \cup N_{B_{j+1}}(v_{j+1})$. So by Case 3 of Algorithm Numbering g , $v_j \notin D$. Note that $g(u_j) < g(w_{j+1})$ or $g(u_{j+1}) < g(w_j)$.

- If $g(u_j) < g(w_{j+1})$, then Algorithm Trees chooses neighbors p_1, p_3 of v_j in B_j such that $T_{u_j}^j[u_j, p_1]$ and $T_{v_{j-1}}^j[v_{j-1}, p_3]$ are disjoint and neighbors

p_2, p_4 of v_j in B_{j+1} such that $T_{w_{j+1}}^{j+1}[w_{j+1}, p_2]$ and $T_{v_{j+1}}^{j+1}[v_{j+1}, p_4]$ are disjoint. By construction, $T_1[u_j, v_j] = T_{u_j}^j[u_j, p_1] + \{v_j, v_j p_1\}$, $T_3[v_{j-1}, v_j] = T_{v_{j-1}}^j[v_{j-1}, p_3] + \{v_j, v_j p_3\}$, $T_2[w_{j+1}, v_j] = T_{w_{j+1}}^{j+1}[w_{j+1}, p_2] + \{v_j, v_j p_2\}$, and $T_4[v_{j+1}, v_j] = T_{v_{j+1}}^{j+1}[v_{j+1}, p_4] + \{v_j, v_j p_4\}$. Moreover, $T_3[v_0, v_{j-1}] \subseteq \bigcup_{l=1}^{j-1} B_l$ and $T_4[v_k, v_{j+1}] \subseteq \bigcup_{l=j+2}^k B_l$. Thus, $T_1[u_j, v_j], T_2[w_{j+1}, v_j], T_3[v_0, v_j]$, and $T_4[v_k, v_j]$ are internally disjoint. Let $z_1 := u_j$ and $z_2 := w_{j+1}$. Clearly, (1)–(3) hold.

- If $g(u_j) \geq g(w_{j+1})$, then $g(u_{j+1}) < g(w_j)$, and Algorithm Trees chooses neighbors p_2, p_3 of v_j in B_j such that $T_{w_j}^j[w_j, p_2]$ and $T_{v_{j-1}}^j[v_{j-1}, p_3]$ are disjoint and neighbors p_1, p_4 of v_j in B_{j+1} such that $T_{u_{j+1}}^{j+1}[u_{j+1}, p_1]$ and $T_{v_{j+1}}^{j+1}[v_{j+1}, p_4]$ are disjoint. Let $z_1 := u_{j+1}$ and $z_2 := w_j$. One can show as in the above paragraph that $T_1[u_{j+1}, v_j], T_2[w_j, v_j], T_3[v_0, v_j]$, and $T_4[v_k, v_j]$ are internally disjoint and (1)–(3) hold.

Case 4. H_i is a triangle G_{i-1} -chain in G .

Let $I(H_i) := \{v_1, v_2, v_3\}$, let $y_1, y_2, y_3 \in V(\bar{G}_i)$ such that $y_1 v_1, y_2 v_2, y_3 v_3 \in E(G)$, and let $v_j x_j$ ($j = 1, 2, 3$) be the legs of H_i . Assume that v_1, v_2, v_3 are labeled so that $g(x_1) < g(x_2) < g(x_3)$.

The proof can be done by inspecting a small number of cases (Case 4 in Algorithm Trees) and using Lemma 6.7 and Case 4 of Algorithm Numbering g and Algorithm Numbering f . For the sake of completeness, we list for each case the choice for z_1, z_2, z_3 , and z_4 . The verification that they satisfy (1)–(3) is straightforward, and we omit it.

- If $f(y_1) < f(y_2)$ and $f(y_1) < f(y_3)$, then let $z_2 := x_3$ and $z_3 := y_1$.
 If $v = v_1$, then let $z_1 := x_1$ and $z_4 := y_2$.
 If $v = v_2$, then let $z_1 := x_2$ and $z_4 := y_2$.
 If $v = v_3$, then let $z_1 := x_2$ and $z_4 := y_3$.
- If $f(y_2) < f(y_1)$ and $f(y_2) < f(y_3)$, then let $z_2 := x_3$ and $z_3 := y_2$.
 If $v = v_1$, then let $z_1 := x_1$ and $z_4 := y_1$.
 If $v = v_2$, then let $z_1 := x_2$ and $z_4 := y_1$.
 If $v = v_3$, then let $z_1 := x_1$ and $z_4 := y_3$.
- If $f(y_3) < f(y_1) < f(y_2)$, then let $z_2 := x_3$ and $z_4 := y_2$.
 If $v = v_1$, then let $z_1 := x_1$ and $z_3 := y_1$.
 If $v = v_2$, then let $z_1 := x_2$ and $z_3 := y_1$.
 If $v = v_3$, then let $z_1 := x_1$ and $z_3 := y_3$.
- If $f(y_3) < f(y_2) < f(y_1)$, then let $z_2 := x_3$ and $z_4 := y_1$.
 If $v = v_1$, then let $z_1 := x_1$ and $z_3 := y_2$.
 If $v = v_2$, then let $z_1 := x_2$ and $z_3 := y_2$.
 If $v = v_3$, then let $z_1 := x_2$ and $z_3 := y_3$.

This completes the proof of Lemma 6.11. \square

LEMMA 6.12. Let $i \in \{1, \dots, t - 1\}$. Then for any $u, v \in D_i$, with $g(u) < g(v)$, $T_1[r, u]$ and $T_2[r, v]$ are internally disjoint paths in G_i .

Proof. We will prove the lemma by induction on i . The basis of induction is $i = 0$ with $D_0 := \{r\}$ and $G_0 := (\{r\}, \emptyset)$. So assume that $i > 0$ and the lemma holds for $i - 1$. We consider the four cases of Algorithm Numbering g .

Case 1. H_i is an elementary G_{i-1} -chain in G .

Let $H_i := v_0 B_1 v_1 B_2 v_2$, with $g(v_0) < g(v_2)$. By (1) of Lemma 6.2, $E(T_1 \cap H_i) = \{v_0 v_1\}$ and $E(T_2 \cap H_i) = \{v_1 v_2\}$. Recall that $D_i = D_{i-1} \cup \{v_1\}$.

If $u, v \in D_{i-1}$ and $g(u) < g(v)$, then by the induction hypothesis, $T_1[r, u]$ and $T_2[r, v]$ are internally disjoint paths in G_{i-1} . Thus, it suffices to prove the following:

for any $u, v \in D_i$, with $g(u) < g(v)$ and $v_1 \in \{u, v\}$, $T_1[r, u]$ and $T_2[r, v]$ are internally disjoint paths in G_i .

Assume first that $u = v_1$. Then $v = v \in D_{i-1}$. Since $g(v_0) < g(v_1) < g(v)$, it follows from the induction hypothesis that $T_1[r, v_0]$ and $T_2[r, v]$ are internally disjoint paths in G_{i-1} . Therefore, $T_1[r, v_1] = T_1[r, v_0] + \{v_1, v_1v_0\}$ and $T_2[r, v]$ are internally disjoint paths in G_i .

Now suppose $v = v_1$. Then $u \in D_{i-1}$. Since $g(u) < g(v_1) < g(v_2)$, it follows from the induction hypothesis that $T_1[r, u]$ and $T_2[r, v_2]$ are internally disjoint paths in G_{i-1} . Therefore, $T_1[r, u]$ and $T_2[r, v_1] = T_2[r, v_2] + \{v_1, v_1v_2\}$ are internally disjoint paths in G_i .

Case 2. $i = 1$, or H_i is an up G_{i-1} -chain in G but not an elementary G_{i-1} -chain in G .

Let $H_i := v_0B_1v_1 \dots v_{k-1}B_kv_k$, with $v_0 = v_k = r$ when $i = 1$, and $g(v_0) < g(v_k)$ when $i \neq 1$. For each 2-connected B_j , let u_j, w_j denote the terminals of B_j^+ other than v_{j-1}, v_j , with $f(u_j) < f(w_j)$, and let $T_{v_{j-1}}^j, T_{v_j}^j, T_{u_j}^j, T_{w_j}^j$ denote the trees rooted, respectively, at v_{j-1}, v_j, u_j, w_j in the independent spanning $\{v_{j-1}, v_j, u_j, w_j\}$ -system of B_j^+ computed in Assumption 4.5. This is the same as in Case 2 of Algorithm Trees. Let $u, v \in D_i$ with $g(u) < g(v)$.

If $u, v \in D_{i-1}$, then by the induction hypothesis, $T_1[r, u]$ and $T_2[r, v]$ are internally disjoint paths in G_{i-1} .

If $u \in D_i - D_{i-1}$ and $v \in D_{i-1}$, then by the construction in Case 2 of Algorithm Numbering g , $g(v_0) < g(u) < g(v)$. By the induction hypothesis, $T_1[r, v_0]$ and $T_2[r, v]$ are internally disjoint paths in G_{i-1} . Since $T_1[v_0, u]$ is a path in $H_i - v_k$ by (1) of Lemma 6.3 when $i = 1$, or by (1) of Lemma 6.4 when $i \neq 1$, $T_1[r, v]$ and $T_2[r, v]$ are internally disjoint paths in G_i .

If $u \in D_{i-1}$ and $v \in D_i - D_{i-1}$, then by the construction in Case 2 of Algorithm Numbering g , $g(u) < g(v) < g(v_k)$. By the induction hypothesis, $T_1[r, u]$ and $T_2[r, v_k]$ are internally disjoint paths in G_{i-1} . Since $T_2[v_k, v]$ is a path in $H_i - v_0$ by (2) of Lemma 6.3 when $i = 1$, or by (2) of Lemma 6.4 when $i \neq 1$, $T_1[r, v]$ and $T_2[r, v]$ are internally disjoint paths in G_i .

So we may assume that $u, v \in D_i - D_{i-1}$. Let g^i denote the function g at the start of iteration i of Algorithm Numbering g (when it examines H_i in Case 2). Recall that for each $j = 1, \dots, k$ the algorithm computes a sequence σ_j as follows. If B_j is 2 connected, then σ_j is a $(T_{v_{j-1}}^j, T_{v_j}^j)$ -ordering of $N_{B_j^+}(\{u_j, w_j\}) - \{v_{j-1}, v_j\}$. If B_j is trivial, then σ_j is the empty sequence. Moreover, the algorithm extends g^i to $\sigma := \sigma_1, v_1, \sigma_2, \dots, v_{k-1}, \sigma_k$ from v_0 and set $D_i := D_{i-1} \cup \{\sigma\}$. Thus, $u, v \in \{\sigma\}$. Note that since $g(u) < g(v)$, u precedes v in the sequence σ .

First, suppose that there exists no $j \in \{1, \dots, k-1\}$ such that $u, v \in \{\sigma_j\}$. Hence, there is some $j \in \{1, \dots, k-1\}$ such that either

- u appears in the sequence $\sigma_1, v_1, \dots, \sigma_j, v_j$ and v appears in the sequence $\sigma_{j+1}, v_{j+1}, \dots, v_{k-1}, \sigma_k$ or
- u appears in the sequence $\sigma_1, v_1, \dots, \sigma_j$ and v appears in the sequence $v_j, \sigma_{j+1}, v_{j+1}, \dots, v_{k-1}, \sigma_k$.

By (1) and (2) of Lemma 6.3 when $i = 1$ or by (1) and (2) of Lemma 6.4 when $i \neq 1$, $T_1[v_0, u]$ and $T_2[v_k, v]$ are internally disjoint paths in H_i , and by the induction hypothesis, $T_1[r, v_0]$ and $T_2[r, v_k]$ are internally disjoint paths in G_{i-1} . Therefore, $T_1[r, u]$ and $T_2[r, v]$ are internally disjoint paths in G_i .

So, we may assume that there exists some $j \in \{1, \dots, k-1\}$ such that u, v are in the sequence σ_j . Since the sequence σ_j is $(T_{v_{j-1}}^j, T_{v_j}^j)$ -ordered and u precedes v in

$\sigma_j, T_{v_{j-1}}^j[v_{j-1}, u]$ and $T_{v_j}^j[v_j, v]$ are disjoint. By the construction in Algorithm Trees, $T_1[v_{j-1}, u] = T_{v_{j-1}}^j[v_{j-1}, u]$ and $T_2[v_j, v] = T_{v_j}^j[v_j, v]$. By (1) and (2) of Lemma 6.3 when $i = 1$, or by (1) and (2) of Lemma 6.4 when $i \neq 1$, $T_1[v_0, v_{j-1}]$ and $T_2[v_k, v_j]$ are internally disjoint paths in H_i . Moreover, by the induction hypothesis, $T_1[r, v_0]$ and $T_2[r, v_k]$ are internally disjoint paths in G_{i-1} . Therefore, $T_1[r, u]$ and $T_2[r, v]$ are internally disjoint paths in G_i .

Case 3. $i = t$, or H_i is a down G_{i-1} -chain in G but not an elementary G_{i-1} -chain in G .

Let $H_i := v_0 B_1 v_1 \dots v_{k-1} B_k v_k$, with $v_0 = v_k = r$ when $i = t$, and $f(v_0) < f(v_k)$ when $i \neq t$. For each 2-connected B_j , let u_j, w_j denote the terminals of B_j^+ other than v_{j-1}, v_j , with $g(u_j) < g(w_j)$, and let $T_{v_{j-1}}^j, T_{v_j}^j, T_{u_j}^j, T_{w_j}^j$ denote the trees rooted, respectively, at v_{j-1}, v_j, u_j, w_j in the independent spanning $\{v_{j-1}, v_j, u_j, w_j\}$ -system of B_j^+ computed in Assumption 4.5. This is the same as in Case 3 of Algorithm Trees. Let $u, v \in D_i$ with $g(u) < g(v)$. Recall that $D_i = D_{i-1} \cup N_{B_1}(v_0) \cup N_{B_k}(v_k)$.

If $u, v \in D_{i-1}$, then by the induction hypothesis, $T_1[r, u]$ and $T_2[r, v]$ are internally disjoint paths in $G_{i-1} \subset G_i$.

If $u \in D_i - D_{i-1}$ and $v \in D_{i-1}$, then $u \in N_{B_1}(v_0) \cup N_{B_k}(v_k)$. By (1) and (3) of Lemma 6.11, there exists $z_1 \in V(G_{i-1})$ such that $g(z_1) < g(u)$ and $V(T_1[z_1, u] - z_1) \subseteq I(H_i)$. Since $z_1, v \in D_{i-1}$ and $g(z_1) < g(u) < g(v)$, it follows from the induction hypothesis that $T_1[r, z_1]$ and $T_2[r, v]$ are internally disjoint paths in G_{i-1} . Therefore, $T_1[r, u]$ and $T_2[r, v]$ are internally disjoint paths in G_i .

If $u \in D_{i-1}$ and $v \in D_i - D_{i-1}$, then $v \in N_{B_1}(v_0) \cup N_{B_k}(v_k)$. By (1) and (3) of Lemma 6.11, there exists $z_2 \in V(G_{i-1})$ such that $g(v) < g(z_2)$ and $V(T_2[z_2, v] - z_2) \subseteq I(H_i)$. Since $z_2, u \in D_{i-1}$ and $g(u) < g(v) < g(z_2)$, it follows from the induction hypothesis that $T_1[r, u]$ and $T_2[r, z_2]$ are internally disjoint paths in G_{i-1} . Therefore, $T_1[r, u]$ and $T_2[r, v]$ are internally disjoint paths in G_i .

So, we need only to prove the case when $u, v \in D_i - D_{i-1}$. Let g^i denote the function g at the start of iteration i of Algorithm Numbering g (when it examines H_i in Case 3). Now we consider the three subcases of Case 3 of Algorithm Numbering g .

Subcase 3.1. $k = 1$ (thus, B_1 is 2-connected).

Since $(B_1^+, v_0, u_1, v_1, w_1)$ is planar and G is 4-connected, $v_0, v_1 \notin N_{B_1}(v_0) \cup N_{B_1}(v_1)$. Hence, in this case, $D_i - D_{i-1} = N_{B_1^+}(\{v_0, v_1\}) - \{u_1, w_1\} = N_{B_1}(v_0) \cup N_{B_1}(v_1)$. Moreover, Algorithm Numbering g produces a $(T_{u_1}^1, T_{w_1}^1)$ -ordering σ of $N_{B_1^+}(\{v_0, v_1\}) - \{u_1, w_1\}$ and extends g^i to σ from u_1 .

Let $u, v \in D_i - D_{i-1}$, with $g(u) < g(v)$. Then both u and v are in the sequence σ , and u precedes v in σ . Since σ is $(T_{u_1}^1, T_{w_1}^1)$ -ordered, $T_{u_1}^1[u_1, u]$ and $T_{w_1}^1[w_1, v]$ are disjoint. By the construction in Case 3 of Algorithm Trees, $T_1[u_1, u] = T_{u_1}^1[u_1, u]$ and $T_2[w_1, v] = T_{w_1}^1[w_1, v]$. By the induction hypothesis, $T_1[r, u_1]$ and $T_2[r, w_1]$ are internally disjoint paths in G_{i-1} . Therefore, $T_1[r, u]$ and $T_2[r, v]$ are internally disjoint paths in G_i .

Subcase 3.2. $k = 2$, and B_1 or B_2 is trivial.

By symmetry we assume that B_2 is trivial (the arguments are analogous if B_1 is trivial). Note that B_1 is 2-connected because H_i is not an elementary G_{i-1} -chain in G . Thus, $D_i - D_{i-1} = N_{B_1}(v_0) \cup \{v_1\}$.

- If v_1 has no neighbor in $V(G_{i-1})$, then Algorithm Numbering g chooses neighbors q_1, q_2, q_3 of v_1 in B_1 such that $T_{u_1}^1[u_1, q_1], T_{v_0}^1[v_0, q_2]$, and $T_{w_1}^1[w_1, q_3]$ are disjoint and then computes a $(T_{u_1}^1 + \{v_1, v_1 q_1\}, T_{w_1}^1 + \{v_1, v_1 q_3\})$ -ordering σ of $N_{B_1}(v_0) \cup \{v_1\}$ in $B_1^+ \cup B_2$ (recall that $(B_1^+ \cup B_2, v_0, u_1, v_2, w_1)$ is planar). Then Algorithm Numbering g extends g^i to σ from u_1 .

Let $u, v \in D_i - D_{i-1}$, with $g(u) < g(v)$. Then both u and v are in the sequence σ , and u precedes v in σ .

Let us consider first the case when $u \neq v_1$ and $v \neq v_1$. Thus, $u, v \in N_{B_1}(v_0)$. Since σ is $(T_{u_1}^1 + \{v_1, v_1q_1\}, T_{w_1}^1 + \{v_1, v_1q_3\})$ -ordered and u precedes v in σ , $T_{u_1}^1[u_1, u]$ and $T_{w_1}^1[w_1, v]$ are disjoint. By construction (Case 3 of Algorithm Trees), $T_1[u_1, u] = T_{u_1}^1[u_1, u]$ and $T_2[w_1, v] = T_{w_1}^1[w_1, v]$. By the induction hypothesis, $T_1[r, u_1]$ and $T_2[r, w_1]$ are internally disjoint paths in G_{i-1} . Therefore, $T_1[r, u]$ and $T_2[r, v]$ are internally disjoint paths in G_i .

Now suppose that $u = v_1$. Since σ is $(T_{u_1}^1 + \{v_1, v_1q_1\}, T_{w_1}^1 + \{v_1, v_1q_3\})$ -ordered and u precedes v in σ , $T_{u_1}^1[u_1, q_1] + \{v_1, v_1q_1\}$ and $T_{w_1}^1[w_1, v]$ are disjoint. By construction (Case 3 of Algorithm Trees), $T_1[u_1, v_1] = T_{u_1}^1[u_1, q_1] + \{v_1, v_1q_1\}$ and $T_2[w_1, v] = T_{w_1}^1[w_1, v]$. By the induction hypothesis, $T_1[r, u_1]$ and $T_2[r, w_1]$ are internally disjoint paths in G_{i-1} . Therefore, $T_1[r, u]$ and $T_2[r, v]$ are internally disjoint paths in G_i .

So assume $v = v_1$. Since σ is $(T_{u_1}^1 + \{v_1, v_1q_1\}, T_{w_1}^1 + \{v_1, v_1q_3\})$ -ordered and u precedes v in σ , $T_{u_1}^1[u_1, u]$ and $T_{w_1}^1[w_1, q_3] + \{v_1, v_1q_3\}$ are disjoint. By construction (Case 3 of Algorithm Trees), $T_1[u_1, u] = T_{u_1}^1[u_1, u]$ and $T_2[w_1, v_1] = T_{w_1}^1[w_1, q_3] + \{v_1, v_1q_3\}$. By the induction hypothesis, $T_1[r, u_1]$ and $T_2[r, w_1]$ are internally disjoint paths in G_{i-1} . Therefore, $T_1[r, u]$ and $T_2[r, v]$ are internally disjoint paths in G_i .

- If v_1 has a neighbor in $V(G_{i-1})$, then Algorithm Numbering g chooses a vertex $x \in (N_G(v_1) \cap V(G_{i-1}))$ with $g^i(x)$ minimum and computes a $(T_{u_1}^1, T_{w_1}^1)$ -ordering σ of $N_{B_1^+}(v_0) - \{u_1, w_1\}$.

Let $u, v \in D_i - D_{i-1}$, with $g(u) < g(v)$.

Let us consider first the case when $u \neq v_1$ and $v \neq v_1$. Then both u and v are in σ , and u precedes v in σ . Since σ is $(T_{u_1}^1, T_{w_1}^1)$ -ordered and u precedes v in σ , $T_{u_1}^1[u_1, u]$ and $T_{w_1}^1[w_1, v]$ are disjoint. By construction (Case 3 of Algorithm Trees), $T_1[u_1, u] = T_{u_1}^1[u_1, u]$ and $T_2[w_1, v] = T_{w_1}^1[w_1, v]$. By the induction hypothesis, $T_1[r, u_1]$ and $T_2[r, w_1]$ are internally disjoint paths in G_{i-1} . Therefore, $T_1[r, u]$ and $T_2[r, v]$ are internally disjoint paths in G_i .

Now suppose that $u = v_1$. Thus, v is in the sequence σ . Recall how Algorithm Numbering g extends g^i in Subcase 3.2 of Algorithm Numbering g .

If $g(x) > g(u_1)$, then $g^i(x) > g^i(u_1)$, and Algorithm Numbering g extends g^i to σ, v_1 from u_1 . But then $g(v) < g(v_1) = g(u)$, contradicting the assumption that $g(u) < g(v)$.

If $g(x) \leq g(u_1)$, then $g^i(x) \leq g^i(u_1)$, and Algorithm Numbering g extends g^i to v_1, σ from x . By construction (Subcase 3.2 of Algorithm Trees with $j = 1$), $xv_1 \in E(T_1)$ and $T_2[w_1, v] = T_{w_1}^1[w_1, v]$. Since $g(x) < g(w_1)$, by the induction hypothesis, $T_1[r, x]$ and $T_2[r, w_1]$ are internally disjoint paths in G_{i-1} . Therefore, $T_1[r, v_1]$ and $T_2[r, v]$ are internally disjoint paths in G_i .

The case $v = v_1$ can be treated analogously ($g(x) \leq g(u_1)$ cannot occur).

Subcase 3.3. $k \geq 3$, or $k = 2$ and both B_1, B_2 are 2-connected.

In this case, $D_i - D_{i-1} = N_{B_1}(v_0) \cup N_{B_k}(v_k)$. Let $u, v \in D_i - D_{i-1}$ with $g(u) < g(v)$.

Let us consider first the case when $u, v \in N_{B_1}(v_0)$. Thus, B_1 is 2-connected, and Algorithm Numbering g (Subcase 3.3) computes a $(T_{u_1}^1, T_{w_1}^1)$ -ordering σ of $N_{B_1^+}(v_0) - \{u_1, w_1\} = N_{B_1}(v_0)$ and extends g^i to σ from u_1 . Thus, $g(u_1) < g(u) < g(v)$. Since σ is $(T_{u_1}^1, T_{w_1}^1)$ -ordered and u precedes v in σ , $T_{u_1}^1[u_1, u]$ and $T_{w_1}^1[w_1, v]$ are disjoint. By construction (Case 3 of Algorithm trees), $T_1[u_1, u] = T_{u_1}^1[u_1, u]$ and

$T_2[w_1, v] = T_{w_1}^1[w_1, v]$. Since $g(u_1) < g(w_1)$ and by the induction hypothesis, $T_1[r, u_1]$ and $T_2[r, w_1]$ are internally disjoint paths in G_{i-1} . Therefore, $T_1[r, u]$ and $T_2[r, v]$ are internally disjoint paths in G_i .

Suppose, now, that $u, v \in N_{B_k}(v_k)$. Then B_k is 2-connected, and Algorithm Numbering g (Subcase 3.3) computes a $(T_{u_k}^k, T_{w_k}^k)$ -ordering of $N_{B_k^+}(v_k) - \{u_k, w_k\} = N_{B_k}(v_k)$ and extends g^i to σ from u_k . Thus, $g(u_k) < g(u) < g(v)$. Since σ is $(T_{u_k}^k, T_{w_k}^k)$ -ordered and u precedes v in σ , $T_{u_k}^k[u_k, u]$ and $T_{w_k}^k[w_k, v]$ are disjoint. By construction (Case 3 of Algorithm Trees), $T_1[u_k, u] = T_{u_k}^k[u_k, u]$ and $T_2[w_k, v] = T_{w_k}^k[w_k, v]$. Since $g(u_k) < g(w_k)$ and by the induction hypothesis, $T_1[r, u_k]$ and $T_2[r, w_k]$ are internally disjoint paths in G_{i-1} . Therefore, $T_1[r, u]$ and $T_2[r, v]$ are internally disjoint paths in G_i .

So we may assume that $u \in N_{B_1}(v_0)$ and $v \in N_{B_k}(v_k)$, or $u \in N_{B_k}(v_k)$ and $v \in N_{B_1}(v_0)$. By symmetry, assume that $u \in N_{B_1}(v_0)$ and $v \in N_{B_k}(v_k)$. We will prove that there exist vertices $z_1, z_2 \in V(G_{i-1})$, with $g(z_1) < g(z_2)$, such that $T_1[z_1, u]$ and $T_2[z_2, v]$ are internally disjoint paths in G , $V(T_1[z_1, u] - z_1) \subseteq I(H_i)$, and $V(T_2[z_2, v] - z_2) \subseteq I(H_i)$.

Consider the following cases for u and B_1 .

- B_1 is 2-connected. Then, by construction in Algorithm Trees, $T_1[u_1, u] = T_{u_1}^1[u_1, u]$, and let $z_1 := u_1$.
- B_1 is trivial. Thus, $u = v_1$. If B_2 is trivial, then by construction in Subcase 3.1 of Algorithm Trees (with $j = 1$), there exists a neighbor p_1 of v_1 in $V(G_{i-1})$ such that $g(p_1)$ is minimum and $p_1v_1 \in E(T_1)$. In this case, let $z_1 := p_1$.

So assume that B_2 is 2-connected.

- If v_1 has no neighbor in $V(G_{i-1})$, then by construction in Subcase 3.3 (with $j = 1$) of Algorithm Trees, there exists a neighbor p_1 of v_1 in B_2 such that $T_1[u_2, v_1] = T_{u_2}^2[u_2, p_1] + \{v_1, v_1p_1\}$. In this case, let $z_1 := u_2$.
- If v_1 has a neighbor in $V(G_{i-1})$, then Algorithm Trees in Subcase 3.3 (with $j = 1$) chooses $x \in N_G(v_1) \cap V(G_{i-1})$ with $g(x)$ minimum. If $g(x) > g(u_2)$, then by construction there exists a neighbor p_1 of v_1 in B_2 such that $T_1[u_2, v_1] = T_{u_2}^2[u_2, p_1] + \{v_1, v_1p_1\}$. In this case, let $z_1 := u_2$. If $g(x) \leq g(u_2)$, then $xv_1 \in E(T_1)$. In this case, let $z_1 := x$.

Consider the analogous cases for v and B_k .

- B_k is 2-connected. Then by construction in Algorithm Trees, $T_2[w_k, v] = T_{w_k}^k[w_k, v]$, and let $z_2 := w_k$.
- B_k is trivial. Thus, $v = v_{k-1}$. If B_{k-1} is trivial, then by construction in Subcase 3.1 of Algorithm Trees (with $j = k - 1$), there exists a neighbor p_2 of v_{k-1} in $V(G_{i-1})$ such that $g(p_2)$ is not minimum and $p_2v_{k-1} \in E(T_2)$. In this case, let $z_2 := p_2$.

So assume that B_{k-1} is 2-connected.

- If v_{k-1} has no neighbor in $V(G_{i-1})$, then by construction in Subcase 3.2 (with $j = k - 1$) of Algorithm Trees, there exists a neighbor p_2 of v_{k-1} in B_{k-1} such that $T_2[w_{k-1}, v_{k-1}] = T_{w_{k-1}}^{k-1}[w_{k-1}, p_2] + \{v_{k-1}, v_{k-1}p_2\}$. In this case, let $z_2 := w_{k-1}$.
- If v_{k-1} has a neighbor in $V(G_{i-1})$, then Algorithm Trees in Subcase 3.2 chooses $x \in N_G(v_{k-1}) \cap V(G_{i-1})$ with $g(x)$ minimum. If $g(x) > g(u_{k-1})$, then $xv_{k-1} \in E(T_2)$. In this case, let $z_2 := x$. If $g(x) \leq g(u_{k-1})$, then by construction there exists a neighbor p_2 of v_{k-1} in B_{k-1} such that $T_2[w_{k-1}, v_{k-1}] = T_{w_{k-1}}^{k-1}[w_{k-1}, p_2] + \{v_{k-1}, v_{k-1}p_2\}$. In this case, let $z_2 := w_{k-1}$ (this is the same as in the previous paragraph).

So $T_1[z_1, u]$ either is contained in B_1^+ , or is contained in B_2^+ , or is induced by a single edge. Hence, $g(z_1) < g(u)$. Similarly, $T_2[z_2, v]$ either is contained in B_k^+ , or is contained in B_{k-1}^+ , or is induced by a single edge. So $g(v) < g(z_2)$. Since $g(u) < g(v)$, $g(z_1) < g(z_2)$.

Note that if $k = 3$, B_2 is 2-connected, and both paths $T_1[z_1, u]$ and $T_2[z_2, v]$ are contained in B_2^+ , then $u = v_1, v = v_2 = v_{k-1}, T_1[u_2, u] = T_{u_2}^2[u_2, p_1] + \{v_1, v_1 p_1\}$ for some neighbor p_1 of v_1 in B_2 , and $T_2[w_2, v] = T_{w_2}^2[w_2, p_2] + \{v_2, v_2 p_2\}$ for some neighbor p_2 of v_2 in B_2 . In this case, since u, v are $(T_{u_2}^2, T_{w_2}^2)$ -ordered, $T_1[u_2, u]$ and $T_2[w_2, v]$ are disjoint.

Therefore, since $k \geq 3$, it is not hard to see that $T_1[z_1, u]$ and $T_2[z_2, v]$ are disjoint paths in $G, V(T_1[z_1, u] - z_1) \subseteq I(H_i)$, and $V(T_2[z_2, v] - z_2) \subset I(H_i)$.

Since $g(z_1) < g(z_2)$, by the induction hypothesis, $T_1[r, z_1]$ and $T_2[r, z_2]$ are internally disjoint paths in G_{i-1} . Therefore, $T_1[r, u]$ and $T_2[r, v]$ are internally disjoint paths in G_i .

Case 4. H_i is a triangle G_{i-1} -chain in G .

By Algorithm Numbering $g, D_i - D_{i-1} = \{v_1, v_2, v_3\}$ and $g(v_1) < g(v_2) < g(v_3)$. Thus, it suffices to show that the following pairs are internally disjoint: $T_1[r, v_1]$ and $T_2[r, v_2], T_1[r, v_2]$ and $T_2[r, v_3],$ and $T_1[r, v_1]$ and $T_2[r, v_3]$. This can be done by inspecting Case 4 of Algorithm Trees. \square

Recall that Algorithm Numbering f with input $\mathcal{C} := (H_1, \dots, H_t)$ computes a numbering f and sets $D'_{t+1}, D'_t, D'_{t-1}, \dots, D'_2$. The next lemma can be proved, analogously to Lemma 6.12. We give only some detail for Case 4, as f and g are not symmetric in that case.

LEMMA 6.13. *Let $i \in \{1, \dots, t\}$. Then for any $u, v \in D'_i$ with $f(u) < f(v)$, $T_3[r, u]$ and $T_4[r, v]$ are internally disjoint paths in \bar{G}_i .*

Proof. We use the notation in the proof of Lemma 6.12 and assume H_i is a triangle G_{i-1} -chain in G . By inspecting Case 4 of Algorithm Numbering f and Algorithm Trees, we have the following.

- If $f(y_1) < f(y_2)$ and $f(y_1) < f(y_3)$, then $f(v_1) < f(v_2) < f(v_3)$. So we can show that $T_3[r, v_1]$ and $T_4[r, v_2]$ are internally disjoint, $T_3[r, v_1]$ and $T_4[r, v_3]$ are internally disjoint, and $T_3[r, v_2]$ and $T_4[r, v_3]$ are internally disjoint.
- If $f(y_2) < f(y_1)$ and $f(y_2) < f(y_3)$, then $f(v_2) < f(v_1) < f(v_3)$. So we can show that $T_3[r, v_2]$ and $T_4[r, v_1]$ are internally disjoint, $T_3[r, v_2]$ and $T_4[r, v_3]$ are internally disjoint, and $T_3[r, v_1]$ and $T_4[r, v_3]$ are internally disjoint.
- If $f(y_3) < f(y_1) < f(y_2)$, then $f(v_3) < f(v_1) < f(v_2)$. So we can show that $T_3[r, v_3]$ and $T_4[r, v_1]$ are internally disjoint, $T_3[r, v_3]$ and $T_4[r, v_2]$ are internally disjoint, and $T_3[r, v_1]$ and $T_4[r, v_2]$ are internally disjoint.
- If $f(y_3) < f(y_2) < f(y_1)$, then $f(v_3) < f(v_2) < f(v_1)$. So we can show that $T_3[r, v_3]$ and $T_4[r, v_2]$ are internally disjoint, $T_3[r, v_3]$ and $T_4[r, v_1]$ are internally disjoint, and $T_3[r, v_2]$ and $T_4[r, v_1]$ are internally disjoint. \square

THEOREM 6.14. *Given a 4-connected graph $G, r \in V(G)$, and a nonseparating chain decomposition $\mathcal{C} := (H_1, \dots, H_t)$ of G rooted at r , Algorithm Trees computes four independent spanning trees rooted at r .*

Proof. By Corollary 6.10, T_1, T_2, T_3, T_4 are spanning trees of G . Let us prove that they are independent with r as root. Let $v \in V(G) - \{r\}$. Suppose that v is an internal vertex of a good chain H_i in the decomposition \mathcal{C} . By Lemma 6.11 there exist $z_1, z_2, z_3, z_4 \in V(G)$ such that

- (i) $z_1, z_2 \in V(G_{i-1})$, and either $g(z_1) < g(z_2)$ or $z_1 = z_2 = r$,
- (ii) $z_3, z_4 \in V(\bar{G}_i)$, and either $f(z_3) < f(z_4)$ or $z_3 = z_4 = r$, and

- (iii) $T_i[z_i, v]$, $i = 1, 2, 3, 4$, are internally disjoint paths and $V(T_i[z_i, v] - z_i) \subseteq I(H_i)$.

By Lemma 6.12, if $g(z_1) < g(z_2)$, then $T_1[r, z_1]$ and $T_2[r, z_2]$ are internally disjoint paths in G_{i-1} . Obviously, the same holds if $z_1 = z_2 = r$. Similarly, by Lemma 6.13, if $f(z_3) < f(z_4)$, then $T_3[r, z_3]$ and $T_4[r, z_4]$ are internally disjoint paths in \bar{G}_i , and the same holds if $z_3 = z_4 = r$. Therefore, $T_1[r, v], T_2[r, v], T_3[r, v]$, and $T_4[r, v]$ are internally disjoint. Hence, T_1, T_2, T_3 , and T_4 are independent spanning trees of G rooted at r . \square

LEMMA 6.15. *Algorithm Trees runs in $O(|V(G)|^3)$ time.*

Proof. By Lemmas 4.6 and 4.7, given \mathcal{C} we can compute numberings g and f in $O(|V(G)|^3)$ time. By Theorem 3.2 we can compute independent spanning systems for all planar sections in \mathcal{C} in $O(|V(G)| + |E(G)|)$ time.

We will show that at each iteration the time spent by Algorithm Trees is $O(|V(G)|^2)$ time. Since the number of iterations is at most $|V(G)|$, this implies the result.

Suppose we are at iteration i of Algorithm Trees.

One can see easily that if Case 1 or Case 4 occurs, then Algorithm Trees uses constant time. Thus, we may assume that Case 2 or Case 3 occurs.

Suppose that Case 2 occurs. The initial updating of T_1, T_2, T_3, T_4 (before Subcases 2.1–2.4 are dealt with) can be done in $O(|V(G)|)$ time. Then for each $j \in \{1, \dots, k\}$ the algorithm inserts v_j into the subgraphs T_1, T_2, T_3, T_4 according to Subcases 2.1–2.4. One can see that Subcase 2.1 can be executed in $O(1)$ time. In the other cases, the algorithm has to solve one of the following problems (at most twice).

- (1) Given a planar graph (B, v', u, v, w) and an independent spanning $\{v', u, v, w\}$ -system $\{T_{v'}, T_u, T_v, T_w\}$ of B (with $T_{v'}, T_u, T_v, T_w$ rooted, respectively, at v', u, v, w), find three neighbors p_1, p_2, p_3 of v in B such that $T_{v'}[v', p_1], T_u[u, p_2]$, and $T_w[w, p_3]$ are disjoint.
- (2) Given a planar graph (B, v', u, v, w) and an independent spanning $\{v', u, v, w\}$ -system $\{T_{v'}, T_u, T_v, T_w\}$ of B (with $T_{v'}, T_u, T_v, T_w$ rooted, respectively, at v', u, v, w), find two neighbors p_1, p_2 of v in B such that $T_{v'}[v', p_1]$ and $T_u[u, p_2]$ are disjoint.

By Lemmas 3.6 and 3.7, both problems can be solved in $O(|V(B)|)$ time. Thus, it is not hard to see that the time spent by Algorithm Trees in Case 2 is $O(|V(G)|^2)$.

Case 3 is analogous to Case 2, and by an argument similar to the last paragraph, one can show that Algorithm Trees uses $O(|V(G)|^2)$ time in this case as well. \square

Now we are almost ready to prove Theorem 1.1, except that if we apply Theorem 2.8 directly to a 4-connected graph G to find a nonseparating chain decomposition of G , we spend $O(|V(G)|^2|E(G)|)$ time. We can obtain an $O(|V(G)|^3)$ algorithm by using the following result of Ibaraki and Nagamochi [10].

THEOREM 6.16. *Let G be a k -connected graph for some integer $k \geq 1$. Then one can find in $O(|V(G)| + |E(G)|)$ time a spanning k -connected subgraph of G with $O(|V(G)|)$ edges.*

Proof of Theorem 1.1. Let G be a 4-connected graph, and let $r \in V(G)$. Apply Theorem 6.16 to G , and let G' be the resulting spanning 4-connected subgraph of G .

Applying Theorem 2.8 to G' , we can find a nonseparating chain decomposition \mathcal{C} of G' in $O(|V(G')|^3)$ time (and hence in $O(|V(G)|^3)$ time).

Finally, apply Theorem 6.14 to G, \mathcal{C} and find four independent spanning trees T_1, T_2, T_3, T_4 of G' rooted at r . By Lemma 6.15, this is done in $O(|V(G')|^3)$ time, and hence in $O(|V(G)|^3)$ time. Clearly, T_1, T_2, T_3, T_4 are independent spanning trees of G rooted at r . \square

Acknowledgment. We thank an anonymous referee for correcting our mistakes and bringing references [6] and [14] to our attention.

REFERENCES

- [1] J. CHERIYAN AND S. N. MAHESHWARI, *Finding nonseparating induced cycles and independent spanning trees in 3-connected graphs*, J. Algorithms, 9 (1988), pp. 507–537.
- [2] T. H. CORMEN, C. E. LEISERSON, R. L. RIVEST, AND S. CLIFFORD, *Introduction to Algorithms*, 2nd ed., McGraw–Hill, Boston, MA, 2001.
- [3] S. CURRAN, *Independent Trees in 4-Connected Graphs*, Ph.D. thesis, Georgia Institute of Technology, Atlanta, GA, 2003.
- [4] S. CURRAN, O. LEE, AND X. YU, *Chain decomposition of 4-connected graphs*, SIAM J. Discrete Math., 19 (2005), pp. 848–880.
- [5] D. DOLEV, J. Y HALPERN, B. SIMONS, AND R. STRONG, *A new look at fault tolerant network routing*, in Proceedings of the 16th Annual ACM Symposium on Theory of Computing, 1984, pp. 526–535.
- [6] A. FRANK, *Connectivity and network flows*, in Handbook of Combinatorics 1, R. L. Graham, M. Grötschel, and L. Lovász, eds., Elsevier, Amsterdam, 1995, pp. 111–177.
- [7] J. E. HOPCROFT AND R. E. TARJAN, *Efficient planarity testing*, J. Assoc. Comput. Mach., 21 (1974), pp. 549–568.
- [8] W.-L. HSU AND W.-K. SHIH, *A new planarity test*, Theoret. Comput. Sci., 223 (1999), pp. 179–191.
- [9] A. HUCK, *Independent trees in graphs*, Graphs Combin., 10 (1994), pp. 29–45.
- [10] T. IBARAKI AND H. NAGAMACHI, *A linear-time algorithm for finding a sparse k -connected spanning subgraph of a k -connected graph*, Algorithmica, 7 (1992), pp. 583–596.
- [11] A. ITAI AND M. RODEH, *The multi-tree approach to reliability in distributed networks*, in Proceedings of the 25th Annual IEEE Symposium on the Foundations of Computer Science, 1984, pp. 137–147.
- [12] A. ITAI AND A. ZEHAVI, *Three tree-paths*, J. Graph Theory, 13 (1989), pp. 175–188.
- [13] K. MIURA, S. NAKANO, T. NISHIZEKI, AND D. TAKAHASHI, *A linear-time algorithm to find four independent spanning trees in four connected planar graphs*, Internat. J. Found. Comput. Sci., 10 (1999), pp. 195–210.
- [14] A. SCHRIJVER, *Problem 32*, in Combinatorial Optimization Volume C, Springer-Verlag, Berlin, 2003, pp. 1456–1457.

EFFICIENT SIMULATIONS BY QUEUE MACHINES*

HOLGER PETERSEN[†] AND JOHN MICHAEL ROBSON[‡]

Abstract. The following simulations by machines equipped with a one-way input tape and additional queue storage are shown:

- Every nondeterministic single-tape Turing machine (no separate input-tape) with time bound $t(n)$ can be simulated by one queue in $O(t(n))$ time.
- Every deterministic machine with a one-turn pushdown store can be simulated deterministically by one queue in $O(n\sqrt{n})$ time.
- Every Turing machine with several multidimensional tapes accepting with time bound $t(n)$ can be simulated by two queues in $O(t(n)\log^2 t(n))$ time.
- Every deterministic Turing machine with several linear tapes accepting with time bound $t(n)$ can be simulated deterministically in time $O(t(n)\log t(n))$ by a queue and a pushdown store.

The first two results appear to be the first subquadratic simulations of other storage devices by one queue.

Key words. pushdown automata, grammars, multiqueue machines, multitape machines, simulation, upper bounds

AMS subject classification. 68Q05

DOI. 10.1137/S0097539799350608

1. Introduction. A classical result, essentially due to Post, says that a machine with a single queue is able to perform any computation a Turing machine can; see, e.g., [12]. Turing machines with auxiliary storage devices like pushdowns or stacks are well studied, and the complexity of simulations between machines with such storages and tapes has been thoroughly investigated. In contrast, fewer results have been obtained for the storage device queue. It is known that one-queue machines can simulate several tapes, pushdowns, and queues with quadratic slowdown [9, Theorem 3.1]. Nondeterministic two-queue machines can simulate any number of queues in linear time [9, Theorem 4.2].

For deterministic devices with several queues Hühne [8] gives a simulation of $t(n)$ time-bounded multistorage Turing machines on $O(t(n)\sqrt[k]{t(n)})$ time-bounded machines with $k \geq 2$ queues. He also reports almost matching lower bounds for online simulations of these storage devices.

Li and Vitányi [10] report lower bounds for simulating one queue on other storages without the online restriction.

In the framework of formal languages, machines with one or more queues have been investigated, e.g., in [16, 2].

Hartmanis and Stearns [5] showed that a k -dimensional tape machine with time bound $t(n)$ could be simulated by a linear tape machine in time $O(t^2(n))$. Pippenger and Fischer [14] improved the time to $O(t^{2-1/k}(n))$, and the result of Hennie [6] (with

*Received by the editors January 15, 1999; accepted for publication (in revised form) September 2, 2005; published electronically March 3, 2006. This research was supported in part by the French-German project PROCOPÉ.

<http://www.siam.org/journals/sicomp/35-5/35060.html>

[†]Universität Stuttgart, Institut für Formale Methoden der Informatik, Universitätsstraße 38, 70569 Stuttgart, Germany (petersen@informatik.uni-stuttgart.de).

[‡]LaBRI, Université Bordeaux 1, 351 cours de la Libération, 33405 Talence Cedex, France (robson@labri.fr).

the correction from [3]) shows that this is optimal, at least for online deterministic simulation. Grigor'ev [4] and Loui [11] showed how to reduce the time when the simulating machine uses m -dimensional tapes ($m > 1$); they used nondeterministic and deterministic machines, respectively. Monien [13] improved the result in case of nondeterministic simulation to use only linear tapes and time $O(t(n) \log^2 t(n))$. We show that this bound also holds for nondeterministic simulation by two queues. Rosenberg [15] presented an $O(n \log n)$ time algorithm for context free language recognition on a machine with two queues. Rosenberg's proof—being based on context free grammars—does not seem to generalize to the simulation of storage devices other than a single pushdown.

We also give a self-contained simulation of multistorage machines by machines with a queue and a pushdown.

2. Preliminaries. We adopt the concepts from [10, 9]. The simulated devices will be introduced below. Unless stated otherwise our simulating machines are nondeterministic and are equipped with a single one-way head on a read-only input tape. The machines can determine end-of-input, have access to one or more first-in-first-out queues storing symbols, and are able to signal acceptance of their input. Depending on the symbols read by the input head and at the front of the queues, a finite control determines one or more of the following operations:

- advance the input head to the next cell,
- dequeue the first symbols of some queues,
- enqueue at most one symbol per queue.

After these operations, control is transferred to the next state.

A machine accepts in time $t(n)$ if, for all accepted inputs of length n , the machine admits a computation that ends with acceptance after at most $t(n)$ steps.

Simulation will be understood in the most general sense; i.e., machine A simulates B if both machines accept the same set of inputs. Note that other concepts of simulation are frequently used, notably step-by-step simulation or simulation of a specified retrieval function of a storage unit (e.g., a two-dimensional array with one point of access which can be moved one row or column at a time).

3. Results.

THEOREM 1. *Every nondeterministic bi-infinite single-tape Turing machine accepting in $t(n)$ steps can be simulated by a nondeterministic one-queue machine in $O(t(n))$ steps.*

Proof. We call the Turing machine to be simulated S and the queue machine Q . Let the tape cells of S be labeled with consecutive integers; the first symbol of the input is labeled with 0. We assume without loss of generality that S moves its head in every step, that there is a single final state, and that the head movement in every step reaching this final state is to the right.

Recall that a crossing sequence at the boundary between two adjacent tape cells consists of the chronological sequence of states to which the finite control of S transfers control as the head crosses the boundary. Here we will denote by c_i the crossing sequence occurring in a computation between cell $i - 1$ and i , and we will also encode the direction when going to state q by \vec{q} (right movement) and \overleftarrow{q} (left movement). We adopt the convention that c_0 starts with the initial state of S moving to the right.

The computation of S in terms of crossing sequences can be divided into three stages:

1. involving cells to the left of the input ($i \leq 0$),
2. involving cells within the input w ($0 < i \leq |w|$),

3. involving cells to the right of the input ($i > |w|$).

Queue machine Q simulates the behavior of S on every tape cell used by S , from left to right, i.e., generally not in chronological sequence. During a cycle corresponding to a tape cell, Q keeps the symbol x currently in the cell in its finite control. The symbol x is initialized with a blank in stage 1 and stage 3 and with the actual input symbol in stage 2. This symbol is available to Q from Q 's own input tape.

The idea of the simulation is to have a crossing sequence c_i on the queue and to guess c_{i+1} (which is separated from c_i by $\$$) in a manner consistent with S 's finite control. More specifically, if the remaining suffix of c_i on the queue is c , we have the following cases, which are nonexclusive (c' is the part of the crossing sequence not affected by the step currently simulated):

- $c = \overrightarrow{q_1} \overleftarrow{q_2} c'$ and there is a transition from q_1 to q_2 reading x , writing some symbol y , and moving the head left. Then Q dequeues $\overrightarrow{q_1} \overleftarrow{q_2}$ and replaces x with y .
- $c = \overrightarrow{q_1} c'$ and there is a transition from q_1 to q_2 reading x , writing some symbol y , and moving the head right. Then Q dequeues $\overrightarrow{q_1}$, enqueues $\overrightarrow{q_2}$, and replaces x with y .
- $c = \overleftarrow{q_2} c'$ and there is a transition from q_1 to q_2 reading x , writing some symbol y , and moving the head left. Then Q dequeues $\overleftarrow{q_2}$, enqueues $\overleftarrow{q_1}$, and replaces x with y .
- There is a transition from q_1 to q_2 reading x , writing some symbol y , and moving the head right. Then Q enqueues $\overleftarrow{q_1} \overrightarrow{q_2}$ and replaces x with y .

If the last symbol of the current crossing sequence has been processed and no further operations according to the last case above occur, the marker symbol $\$$ is dequeued, enqueued, and the next cycle starts.

Should the final state be reached, then no successor state is stored on the queue, but the fact that it has been encountered is recorded in the finite control of Q .

The simulation is initiated by guessing zero or more pairs of states according to the last case to be inserted into the queue with the initial tape symbol being a blank. It proceeds in stage 1 until Q guesses that tape cell -1 has just been processed, c_{-1} is stored on the queue, and stage 2 of the simulation is about to start. At this moment, S 's initial state is inserted into the queue as the first element of c_0 . After c_0 has been assembled, the input is read in every cycle, until the last symbol is consumed and $c_{|w|}$ has been formed. The simulation continues in stage 3 until the queue contains no symbol except $\$$. The machine Q eventually accepts when S 's final state has been encountered during the simulation.

Suppose S accepts input w . We fix an accepting computation C on w using tape cells only between positions ℓ and r . To each cell i with $\ell + 1 \leq i \leq r$ we associate the crossing sequence c_i arising from C . By its definition, Q can first guess $c_{\ell+1}$ and then inductively generate c_{i+1} from c_i in a nondeterministic way. When stage 2 starts, Q begins to read input symbols until stage 3 starts. In this way Q can execute an accepting computation (the final state of S is encountered, since C is accepting).

Conversely, if Q accepts an input w , then the contents of the queue after a full cycle of the accepting computation of Q can be assembled into an accepting computation of S . This is done by starting at the step when Q reads the first symbol of w and picking the first entry of the queue. Then we follow the arrows and always choose "fresh" queue entries, resulting in an accepting computation of S .

The number of steps executed by S is equal to the sum of the lengths of all crossing sequences. For every element of a crossing sequence, Q executes a number of

steps bounded by a constant. This shows the claimed time bound. \square

A converse of the above simulation is not possible, showing that queue machines are stronger than single-tape Turing machines.

OBSERVATION 1. *The language $D^\# = \{w\#w \mid w \in \{0,1\}^*\}$ can be accepted in real time by a deterministic one-queue machine but not by any nondeterministic single-tape Turing machine working in $o(n^2)$ time.*

Proof. Techniques due to Hennie and Barzdin show that $D^\#$ cannot be accepted in $o(n^2)$ time by a single-tape machine; see [17, Theorem 8.13]. On the other hand, a queue machine stores all symbols up to the first $\#$ on the queue and compares them to the string following $\#$. The input is rejected if a mismatch is detected or no separator $\#$ is found; otherwise it is accepted. \square

In the following we present a simulation of a pushdown automaton by a machine with a single queue. It will help to first recall the trivial simulation of one pushdown store by a queue, a special case of [9, Theorem 3.1]. Every push operation can be simulated in constant time by enqueueing a symbol, while pop operations are simulated by cycling through the entire queue contents, locating the top symbol, and removing it. Since the size of the queue is linearly bounded by the number of operations being simulated, this results in a quadratic overhead. Notice that after simulating one pushdown operation, this trivial approach encodes the pushdown contents verbatim. The key to a more efficient simulation is a different encoding and processing of the information stored.

The next machine to be simulated has a one-turn pushdown. (In any computation the machine may switch from pushing to popping at most once.) Machines in this class accept exactly the deterministic linear context free languages.

THEOREM 2. *Every deterministic machine with a one-turn pushdown store can be simulated by a deterministic machine with one queue in $O(n\sqrt{n})$ time.*

Proof. Let P be a deterministic one-turn pushdown machine. Note that if P accepts, it does so in linear time.

Let Q be the queue machine with queue alphabet $X \cup \{\#, *, \$\}$, where X is the pushdown alphabet of P , the union is disjoint, and the length of the input is n .

The idea of the simulation is to divide Q 's computation into three stages. In the first stage, Q simulates push operations of P by writing the pushed symbols into its queue in the same way as in the trivial simulation outlined above. When P switches to popping the pushdown contents, the approach deviates from the trivial one.

After stage 1, queue machine Q suspends the simulation and prepares its storage in order to speed up the access to pushdown symbols. The main observation is that each section of the queue contents eventually has to be read in a reversed fashion. Therefore we would like to form mirror images of the sections. This will be fast if the sections are short since we can process one symbol of *each* section in one round, reading the queue contents once. On the other hand, if the sections are short, there will be many of them. Because the sections are reversed as well, access time during stage 3 depends on exactly how many. A compromise between speed of preprocessing and access leads to a size of the sections proportional to \sqrt{n} .

In stage 3 of the simulation, Q simulates the pop operations of P , from time to time rearranging the queue contents. A more detailed description of the second and third stages follows.

Suppose that v is the queue contents of Q when P reverses its access to the pushdown and stage 2 starts, $|v| \in O(n)$. First Q divides v into strings v_i with $|v_i| \in \Theta(\sqrt{|v|})$ except for the last string, which may be shorter. To do this, Q marks

the end of the queue with \$ and in one pass writes the string *# after every symbol from X . As long as the number of *'s in the queue exceeds one, Q deletes every second * starting with the first one, and in every second pass also deletes every second #, except the one immediately before \$. Thus Q makes $\lfloor \log |v| \rfloor$ passes in $O(|v| \log |v|)$ steps and $\lfloor \lfloor \log |v| \rfloor / 2 \rfloor$ times approximately divides the number of #'s by 2. After these operations, v is divided into k blocks v_i terminated by #. We have

$$|v_i| \leq 2^{\lfloor \frac{\lfloor \log |v| \rfloor}{2} \rfloor} \leq \sqrt{|v|}$$

and

$$k \leq \frac{|v|}{2^{\lfloor \frac{\lfloor \log |v| \rfloor}{2} \rfloor}} \leq 2\sqrt{2}\sqrt{|v|}.$$

Next, Q in one pass inserts a symbol * before every #. Then it starts to reverse the blocks v_i by deleting a symbol $x \in X$ from the beginning of each block that is not yet completely reversed, keeping x in its finite control and inserting it after the * in the same block. This process is repeated until all blocks start with *, and then the *'s are deleted in one pass. Each block $v_i\#$ has been transformed into $v_i^R\#$ in a total of $O(|v|\sqrt{|v|})$ steps.

The third stage of Q 's operation requires a preparation that speeds up Q 's access to the last block on the queue. In k cycles Q inserts a * in front of every block that already contains a * and of the first block that has not received a * up to that point, until every block has received at least one *. The effect of these operations is that the blocks contain $k, k-1, \dots, 1$ symbols *. The preparation is $O(n\sqrt{n})$ time bounded. Now Q enters the third stage and simulates P 's pushdown-reading operations by repeatedly rotating blocks to the rear of the queue until it finds the unique block with a single *. It deletes * and reads this block of pushdown symbols while processing the next input segment until it encounters the trailing #, deletes this symbol, and rewrites \$ at the rear of the queue. Then in one cycle it deletes a single * in every block. It repeats this sequence of operations until the input is exhausted. Emptiness of the pushdown store can easily be detected since then the \$ is the first symbol in the queue. Each rotation takes $O(|v|)$ time, and there are $k \in O(\sqrt{|v|})$ blocks; therefore this stage is $O(|v|\sqrt{|v|})$ time-bounded. \square

We remark that the previous simulation applies to the language $L = \{w\#w^R \mid w \in \{0,1\}^*\}$ investigated in [9, section 3.2]. Our upper bound almost matches the lower bound $\Omega(n^{4/3}/\log n)$ from [9].

The proof of the next result uses ideas from [13].

THEOREM 3. *Every nondeterministic Turing machine with several multidimensional work tapes accepting with time bound $t(n)$ can be simulated by two queues in $O(t(n)\log^2 t(n))$ time.*

Proof. For convenience we will describe a simulator that is equipped with a large number of queues. The linear-time simulation of machines with several queues with two queues [9, Theorem 4.2] will give the result.

Let the m work tapes of machine M that is to be simulated be d -dimensional. For tape i our simulator Q has $d+1$ queues. Queue $i(d+1)$ records the read-write operations on tape i ; queues $i(d+1)+1$ through $i(d+1)+d$ contain binary counters that store the distance of M 's head from its initial head position on tape i . More precisely, for a distance k , the reversal of the binary representation of k is written into the corresponding queue followed by a separator symbol #. For each counter,

Q 's finite control records whether the stored value is positive or negative. All counters are initially zero.

The simulation of M is divided into stages. In the first stage Q guesses step-by-step a computation of M , reading input symbols if necessary, and guessing a corresponding step of M . Let the symbol read by this step on tape i be x_i and the symbol written be y_i . The current distances for tape i are k_1, \dots, k_d . Then Q writes a record containing x_i, y_i , and k_1, \dots, k_d (including signs) into queue $i(d+1)$. The distances are copied by rotating the binary representations stored in queues $i(d+1)+1, \dots, i(d+1)+d$. Now Q updates the distances as indicated by the head move on tape i by adding or subtracting one if necessary. These operations are carried out for every tape and take $O(t(n) \log t(n))$ time. If eventually the simulation reaches an accepting state of M , the second stage is started.

In the second stage the consistency of the guessed computation is checked. For every tape i the simulator uses queues $i(d+1)$ and $i(d+1)+1$ for sorting the records according to distances in a stable way. A suitable method is to use radix sort on their binary representations, starting with the least significant bits and marking off used bits. First a new marker is appended to queue $i(d+1)$. Records containing 0 at the current position are put into queue $i(d+1)+1$; the others are moved to the rear of queue $i(d+1)$. If the marker is encountered, the queues are appended, and the next bit position is considered. In case all digits of a number are exhausted while there are still bits to be processed, the symbol # is interpreted as a string of leading zeros. If all bits have been handled, a final pass sorts according to signs. Sorting is done for all dimensions. Then Q checks for every run of records with equal distances that the first symbol read is a blank and that the symbol written by record j is equal to the symbol read by record $j+1$. If it detects an inconsistency, it aborts the simulation; otherwise it accepts.

The number of records is $O(t(n))$, the length of every record is $O(\log t(n))$, and the total number of passes is $O(\log t(n))$. We obtain the required time bound $O(t(n) \log^2 t(n))$. \square

We remark that tapes can simulate queues in linear time; see [17, Lemma 19.9].

Hühne [8] pointed out that his deterministic simulation of multistorage Turing machines in time $O(t(n)\sqrt{t(n)})$ can be performed on a deterministic machine with a queue and a pushdown store. He also mentions an $\Omega(t(n)\sqrt[4]{\log t(n)})$ lower bound.

In the nondeterministic case a queue and a pushdown simulate any number of pushdown stores (and hence tapes) in linear time by adapting the following technique of Book and Greibach [1]. Guess a sequence of partial configurations containing the state of the simulated machine, the topmost symbols of each pushdown store, the input symbol currently scanned, and the operations on input head and storage. This sequence is written onto the queue. Then the simulator checks that the sequence corresponds to a valid computation for each of the pushdown stores and the input.

We give a deterministic simulation of an arbitrary number of tapes on a queue and a pushdown store that almost matches the lower bound.

THEOREM 4. *Any deterministic $t(n)$ -time-bounded multitape Turing machine can be simulated by a deterministic Turing machine accessing a queue and a pushdown store which is $O(t(n) \log t(n))$ -time-bounded.*

Proof. We first note that a tape can be simulated by two pushdown stores with a linear overhead in a straightforward manner. We may thus restrict our attention to an efficient simulation of several pushdown storages with the help of a single pushdown and a queue.

The idea of the simulation is to store the contents of several pushdown stores on separate tracks of the single pushdown store of the simulator. Each storage cell on these tracks may be empty or contain information stored in the simulated pushdown. The concatenation of all nonempty cells on track k from top to bottom constitutes the string stored on the k th simulated pushdown. We point out that the distribution of empty cells may vary between these tracks, depending on the sequence of accesses to the pushdown stores. Bottom markers of the simulated pushdowns are treated in the same way as other symbols in the pushdown alphabet.

In order to simplify the description of our simulation we concentrate on one pushdown and describe the modification of the track corresponding to this pushdown during the simulation of pop and push operations. The contents of other tracks are left unchanged, except for the introduction of additional empty cells above the bottom marker of the simulator.

Each track is divided into a special topmost cell and a potentially infinite sequence of frames of increasing size, where frame i contains 2^{i+1} cells. Each frame is divided into two blocks of equal size. We assume that this structure is easily recognizable, e.g., by marking the borders between frames and blocks. An invariant maintained by the phases of the simulation is that a frame either is empty, has block 0 empty and block 1 full, or has both blocks full. The number of frames is the same for each track, and the bottom symbol of the simulator's pushdown marks the last frame.

The simulation starts with an empty pushdown, except for the topmost symbol and the bottom marker, and an empty queue.

We will first describe the simulation of a pop. Let the i th frame be the first nonempty one; recall that it has block 1 full, and block 0 may be full or empty. The state of the pushdown is described by Figure 1.

Phase 1 (unloading). The simulator unloads the pushdown onto the queue until it has read the first nonempty frame. We assume that the structure imposed on the pushdown is still visible after unloading.

Phase 2 (redistribution). In this phase the simulator uses the pushdown as a scratch memory, separating the contents of the pushdown at the start of this phase from the operations carried out within the phase. It can do so by storing a marker on top of the pushdown. In this way the contents of the pushdown at the end of the redistribution phase are identical to the contents at the start of the phase.

The current state of the queue before the redistribution phase is shown in Figure 2.

First the simulator rotates the queue until it reaches the string previously stored in the nonempty frame and copies the contents of block 0, if it is nonempty, or else block 1 onto the pushdown. Then in another round it overwrites this portion of the queue with the mirror-image of its initial contents, emptying the pushdown (see Figure 3).

The simulator again copies the reversed string onto the pushdown, this time erasing the cells read. Then it puts one symbol into the position of the first cell on the queue and, now rotating the queue contents, 2^i symbols from the top of the pushdown into the first blocks of each of the frames currently on the queue. Since $2^i = 1 + \sum_{k=0}^{i-1} 2^k$, there are just enough cells to accommodate all symbols and completely empty the pushdown by putting 2^k symbols into block 1 of frame k . Each block receiving symbols from the pushdown is full. Therefore the invariant remains true (see Figure 4).

Finally the simulator transforms the entire track into its mirror-image.

Phase 3 (loading). The simulator removes the marker that separated the sections

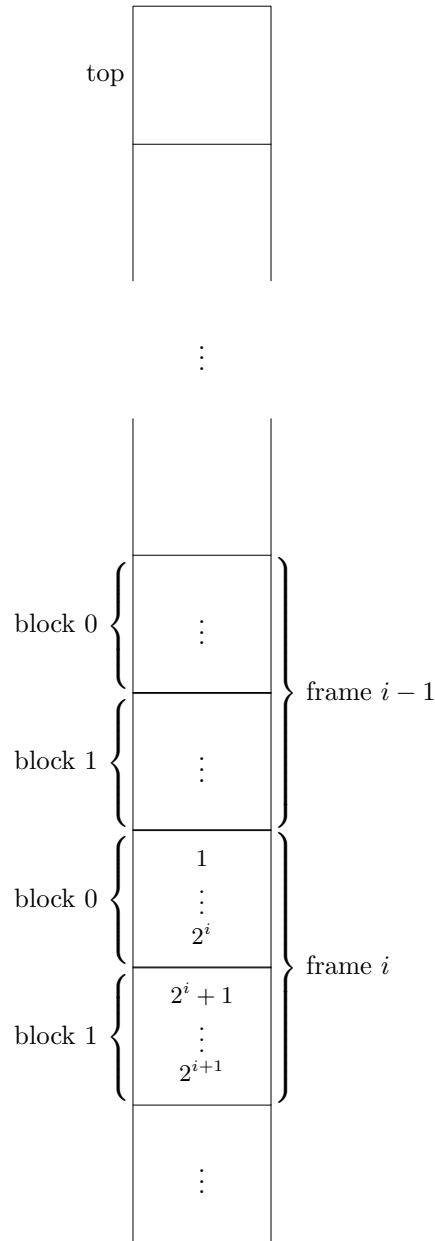


FIG. 1.

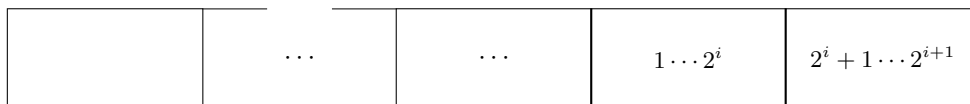


FIG. 2.

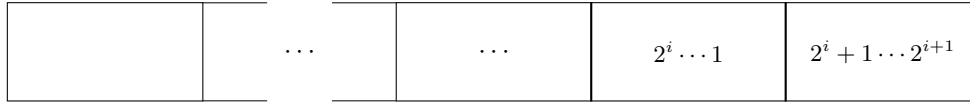


FIG. 3.

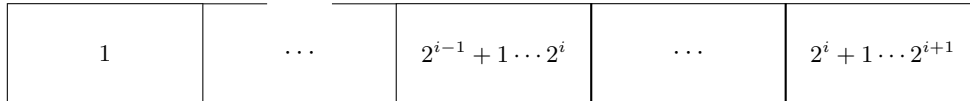


FIG. 4.

not involved in the simulation of the pop operation from the pushdown. Then it empties the queue and stores its contents onto the pushdown. Note that the initial sequence of frames is restored by forming the mirror-image of the contents of the queue at the end of the previous phase. We thus obtain the situation shown in Figure 5.

This completes the simulation of a pop operation.

For a push operation the situation is slightly more complicated, since the current number of frames stored on the pushdown might not suffice for storing another symbol. In the following we describe the simulation of a push operation again in three phases, referring to the corresponding phases in a pop operation if there are common sections.

Phase 1 (unloading). The simulator checks whether the topmost cell is empty. If so, it fills this position with the new symbol and skips the remaining phases. Otherwise it unloads the pushdown onto the queue until it has read a frame that is not completely full or the pushdown becomes empty. In the latter case it allocates a new frame on the queue. This is done by counting the current length ℓ of the queue on the pushdown and adding $\ell + 1$ (for each track) empty cells.

Phase 2 (redistribution). The remaining contents of the pushdown are protected from the forthcoming operations in the same way as described in the simulation of pop operations. Let frame i be the one that is not completely full. The simulator collects the $2^{i+1} - 1$ symbols from the topmost cell and the frames that are full and reverses this string as described above. Then it enters 2^k symbols into block 1 of frame k for each $0 \leq k < i$ and the remaining 2^i symbols into block 1 of frame i if it is empty. Otherwise it uses block 0. Finally the new symbol is written into the topmost cell.

After each operation involving frame i there are 2^i symbols and $2^i - 1$ empty cells above this frame in the pushdown. At most one access to frame i can thus occur within 2^i steps. The number of operations in the simulation described above is proportional to the maximum length of the frames involved. We can therefore bound the time complexity of simulating $t(n)$ steps of the multitape machine by

$$\sum_{k=0}^{\lfloor \log t(n) \rfloor} c2^k \cdot \lfloor t(n)/2^k \rfloor = O(t(n) \log t(n))$$

for some constant factor c . Since the number of tracks is fixed for a given machine, the time complexity of the simulation is also $O(t(n) \log t(n))$. \square

The simulation employed in the preceding proof is based on the idea of using memory blocks of exponentially increasing size from the multitape simulation due to

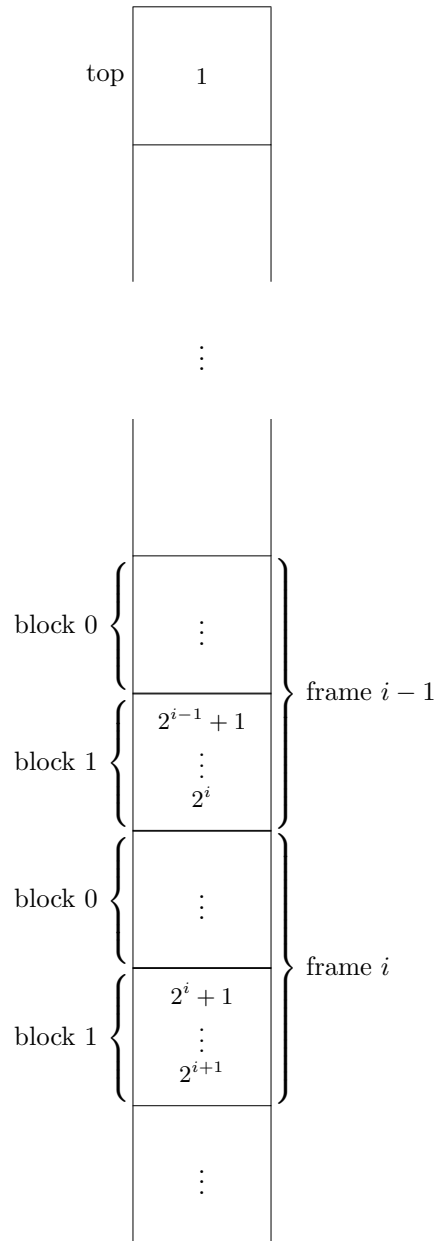


FIG. 5.

Hennie and Stearns [7]. It is known that the latter simulation can be done with a tape and a pushdown store, and the same is true for our simulation by replacing the queue by a tape and modifying the appropriate steps in the simulation. A direct adaption of the simulation from [7] to a machine with queue and pushdown does not, however, seem to be possible, since keeping the majority of memory accesses close to the “home” square in that simulation does not work if the information is stored on a queue. Instead the entire contents would have to be rotated, resulting in a quadratic slow-down.

4. Open problems. A gap remains between the upper bound $O(n\sqrt{n})$ from Theorem 2 and the lower bound $\Omega(n^{4/3}/\log n)$ from [9] for the simulation of a push-down by one queue. The optimality of the bounds in Theorems 3 and 4 are further open problems.

Acknowledgments. The first author would like to thank Jeff Shallit and Pierre McKenzie for comments on an earlier draft of this paper. We also thank Franz-Josef Brandenburg and the anonymous referees for useful remarks.

REFERENCES

- [1] R. V. BOOK AND S. A. GREIBACH, *Quasi-realtime languages*, Math. Systems Theory, 4 (1970), pp. 97–111.
- [2] F.-J. BRANDENBURG, *Multiple equality sets and Post machines*, J. Comput. System Sci., 21 (1980), pp. 292–316.
- [3] D. YU. GRIGOR'EV, *Imbedding theorems for Turing machines of different dimensions and Kolmogorov's algorithms*, Dokl. Akad. Nauk SSSR, 234 (1977), pp. 15–18 (in Russian); English translation in Soviet Math. Dokl., 18 (1977), pp. 588–592.
- [4] D. YU. GRIGOR'EV, *Time complexity of multidimensional Turing machines*, Zapiski Nauchnykh Seminarov Leningradskogo Otdel'niya Matematicheskogo Instituta im. V. A. Steklova AN SSSR, 88 (1979), pp. 47–55 (in Russian); English translation in J. Soviet Math., 20 (1982), pp. 2290–2295.
- [5] J. HARTMANIS AND R. E. STEARNS, *On the computational complexity of algorithms*, Trans. Amer. Math. Soc., 117 (1965), pp. 285–306.
- [6] F. C. HENNIE, *On-line Turing machine computations*, IEEE Trans. Electronic Computers, EC-15 (1966), pp. 35–44.
- [7] F. C. HENNIE AND R. E. STEARNS, *Two-tape simulation of multitape Turing machines*, J. ACM, 13 (1966), pp. 533–546.
- [8] M. HÜHNE, *On the power of several queues*, Theoret. Comput. Sci., 113 (1993), pp. 75–91.
- [9] M. LI, L. LONGPRÉ, AND P. VITÁNYI, *The power of the queue*, SIAM J. Comput., 21 (1992), pp. 697–712.
- [10] M. LI AND P. M. B. VITÁNYI, *Tape versus queue and stacks: The lower bounds*, Inform. and Comput., 78 (1988), pp. 56–85.
- [11] M. C. LOUI, *Simulations among multidimensional Turing machines*, Theoret. Comput. Sci., 21 (1982), pp. 145–161.
- [12] Z. MANNA, *Mathematical Theory of Computation*, McGraw-Hill, New York, 1974.
- [13] B. MONIEN, *About the derivation languages of grammars and machines*, in Proceedings of the 4th International Colloquium on Automata, Languages and Programming (ICALP), Turku, 1977, Lecture Notes in Comput. Sci. 52, M. Steinby, ed., Springer, Berlin, Heidelberg, New York, 1977, pp. 337–351.
- [14] N. PIPPENGER AND M. J. FISCHER, *Relations among complexity measures*, J. ACM, 26 (1979), pp. 361–381.
- [15] B. ROSENBERG, *Fast nondeterministic recognition of context-free languages using two queues*, Inform. Process. Lett., 67 (1998), pp. 91–93.
- [16] R. VOLLMAR, *Über einen Automaten mit Pufferspeicherung (On an automaton with buffer-tape)*, Computing, 5 (1970), pp. 57–70 (in German).
- [17] K. WAGNER AND G. WECHSUNG, *Computational Complexity*, Math. Appl., D. Reidel, Dordrecht, The Netherlands, 1986.

THE COMPLEXITY OF THE LOCAL HAMILTONIAN PROBLEM*

JULIA KEMPE[†], ALEXEI KITAEV[‡], AND ODED REGEV[§]

Abstract. The k -LOCAL HAMILTONIAN problem is a natural complete problem for the complexity class QMA, the quantum analogue of NP. It is similar in spirit to MAX- k -SAT, which is NP-complete for $k \geq 2$. It was known that the problem is QMA-complete for any $k \geq 3$. On the other hand, 1-LOCAL HAMILTONIAN is in P and hence not believed to be QMA-complete. The complexity of the 2-LOCAL HAMILTONIAN problem has long been outstanding. Here we settle the question and show that it is QMA-complete. We provide two independent proofs; our first proof uses only elementary linear algebra. Our second proof uses a powerful technique for analyzing the sum of two Hamiltonians; this technique is based on perturbation theory and we believe that it might prove useful elsewhere. Using our techniques we also show that adiabatic computation with 2-local interactions on qubits is equivalent to standard quantum computation.

Key words. quantum computation, local Hamiltonian problem, complete problems, adiabatic computation

AMS subject classifications. 81P68, 68Q17

DOI. 10.1137/S0097539704445226

1. Introduction. Quantum complexity theory has emerged alongside the first efficient quantum algorithms in an attempt to formalize the notion of an *efficient* algorithm. In analogy to classical complexity theory, several new quantum complexity classes have appeared. A major challenge today consists in understanding their structure and the interrelation between classical and quantum classes.

One of the most important classical complexity classes is NP—nondeterministic polynomial time. This class comprises languages that can be *verified* in polynomial time by a deterministic verifier. The celebrated Cook–Levin theorem (see, e.g., [17]) shows that this class has *complete* problems. More formally, it states that SAT is NP-complete; i.e., it is in NP and any other language in NP can be reduced to it with polynomial overhead. In SAT we are given a set of clauses (disjunctions) over n variables and asked whether there is an assignment that satisfies all clauses. One can consider the restriction of SAT in which each clause consists of at most k literals.

*Received by the editors July 20, 2004; accepted for publication (in revised form) June 23, 2005; published electronically March 3, 2006. A preliminary version of this paper appeared in *Proceedings of the 24th Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS)*, 2004.

<http://www.siam.org/journals/sicomp/35-5/44522/html>

[†]CNRS & LRI, Bat. 490, Université de Paris-Sud, 91405 Orsay Cedex, France, and UC Berkeley, Berkeley, CA 94720. This author’s work was supported by ACI Sécurité Informatique, 2003-n24; projet “Réseaux Quantiques,” ACI-CR 2002-40 and EU 5th framework program RESQ IST-2001-37559; by DARPA and Air Force Laboratory, Air Force Materiel Command, USAF, under agreement F30602-01-2-0524; by DARPA and the Office of Naval Research under grant FDN-00014-01-1-0826; and during a visit supported in part by the National Science Foundation under grant EIA-0086038 through the Institute for Quantum Information at the California Institute of Technology.

[‡]Departments of Physics and Computer Science, California Institute of Technology, Pasadena, CA 91125. This author’s work was supported in part by the National Science Foundation under grant EIA-0086038.

[§]Department of Computer Science, Tel-Aviv University, Tel-Aviv 69978, Israel. This author’s work was supported by an Alon Fellowship, the Binational Science Foundation, the Israel Science Foundation, and the Army Research Office grant DAAD19-03-1-0082 and partly supported by ACI Sécurité Informatique, 2003-n24, projet “Réseaux Quantiques.” Part of this author’s work was carried out during a visit at LRI, Université de Paris-Sud, and he thanks his hosts for their hospitality.

This is known as the k -SAT problem. It is known that 3-SAT is still NP-complete while 2-SAT is in P, i.e., has a polynomial time solution. We can also consider the MAX- k -SAT problem: here, given a k -SAT formula and a number m we are asked whether there exists an assignment that satisfies at least m clauses. It turns out that MAX-2-SAT is already NP-complete; MAX-1-SAT is clearly in P.

The class QMA is the quantum analogue of NP in a probabilistic setting, i.e., the class of all languages that can be probabilistically verified by a quantum verifier in polynomial time (the name is derived from the classical class MA, which is the randomized analogue of NP). This class, which is also called BQNP, was first studied in [13, 12]; the name QMA was given to it by Watrous [20]. Several problems in QMA have been identified [20, 12, 9]. For a good introduction to the class QMA, see the book by Kitaev, Shen, and Vyalvi [12] and the paper by Watrous [20].

Kitaev, inspired by ideas due to Feynman, defined the quantum analogue of the classical SAT problem, the LOCAL HAMILTONIAN problem [12].¹ An instance of k -LOCAL HAMILTONIAN can be viewed as a set of local constraints on n qubits, each involving at most k of them. We are asked whether there is a state of the n qubits such that the expected number of violated constraints is either below a certain threshold or above another, with a promise that one of the two cases holds and both thresholds are at least a constant apart. More formally, we are to determine whether the *groundstate* energy of a given k -local Hamiltonian is below one threshold or above another.

Kitaev proved [12] that the 5-LOCAL HAMILTONIAN problem is QMA-complete. Later, Kempe and Regev showed that even 3-LOCAL HAMILTONIAN is complete for QMA [11]. In addition, it is easy to see that 1-LOCAL HAMILTONIAN is in P. The complexity of the 2-LOCAL HAMILTONIAN problem was left as an open question in [2, 21, 11, 7]. It is not hard to see that the k -LOCAL HAMILTONIAN problem contains the MAX- k -SAT problem as a special case.² Using the known NP-completeness of MAX-2-SAT, we obtain that 2-LOCAL HAMILTONIAN is NP-hard; i.e., any problem in NP can be reduced to it with polynomial overhead. But is it also QMA-complete? Or perhaps it lies in some intermediate class between NP and QMA? Some special cases of the problem were considered by Bravyi and Vyalvi [7]; however, the question still remained open.

In this paper we settle the question of the complexity of 2-LOCAL HAMILTONIAN and show that the following theorem holds.

THEOREM 1.1. *The 2-LOCAL HAMILTONIAN problem is QMA-complete.*

In [12] it was shown that the k -LOCAL HAMILTONIAN problem is in QMA for any constant k (and in fact even for $k = O(\log n)$, where n is the total number of qubits). Hence, our task in this paper is to show that any problem in QMA can be reduced to the 2-LOCAL HAMILTONIAN problem with a polynomial overhead. We give two self-contained proofs for this.

Our first proof is based on a careful selection of gates in a quantum circuit and several applications of a lemma called the *projection lemma*. The proof is quite involved; however, it uses only elementary linear algebra and hence might appeal to some readers.

¹For a survey of the LOCAL HAMILTONIAN problem, see [2].

²The idea is to represent the n variables by n qubits and represent each clause by a Hamiltonian. Each Hamiltonian is diagonal and acts on the k variables that appear in its clause. It “penalizes” the assignment that violates the clause by increasing its eigenvalue. Therefore, the lowest eigenvalue of the sum of the Hamiltonians corresponds to the maximum number of clauses that can be satisfied simultaneously.

Our second proof is based on perturbation theory—a collection of techniques that are used to analyze sums of Hamiltonians. This proof is more mathematically involved. Nevertheless, it might give more intuition as to why the 2-LOCAL HAMILTONIAN problem is QMA-complete. Unlike the first proof, which shows how to represent any QMA circuit by a 2-local Hamiltonian, the second proof shows a reduction from the 3-LOCAL HAMILTONIAN problem (which is already known to be QMA-complete [11]) to the 2-LOCAL HAMILTONIAN problem. To the best of our knowledge, this is the first reduction *inside* QMA (i.e., not from the circuit problem). This proof involves what is known as *third order* perturbation theory (interestingly, the projection lemma used in our first proof can be viewed as an instance of *first order* perturbation theory). We are not aware of any similar application of perturbation theory in the literature and we hope that our techniques will be useful elsewhere.

Adiabatic computation. It has been shown in [3] that the model of adiabatic computation with 3-local interactions is equivalent to the standard model of quantum computation (i.e., the quantum circuit model).³ We strengthen this result by showing that 2-local interactions suffice.⁴ Namely, the model of adiabatic computation with 2-local interactions is equivalent to the standard model of quantum computation. We obtain this result by applying the technique of perturbation theory, which we develop in the second proof of the main theorem.

Recent work. After a preliminary version of our paper appeared [10], Oliveira and Terhal [16] generalized our results and have shown that the 2-LOCAL HAMILTONIAN problem remains QMA-complete even if the Hamiltonians are restricted to nearest neighbor interactions between qubits on a two-dimensional grid. Similarly, they show that the model of adiabatic computation with 2-local Hamiltonians between nearest neighbor qubits on a two-dimensional grid is equivalent to standard quantum computation. Their proof applies the perturbation theory techniques that we develop in this paper and introduces several novel “perturbation gadgets” akin to our three-qubit gadget in section 6.2.

Structure. We start by describing our notation and some basics in section 2. Our first proof is developed in sections 3, 4, and 5. The main tool in this proof, which we name the projection lemma, appears in section 3. Using this lemma, we rederive in section 4 some of the previously known results. Then we give the first proof of our main theorem in section 5. In section 6 we give the second proof of our main theorem. This proof does not require the projection lemma and is in fact independent of the first proof. Hence, some readers might choose to skip sections 3, 4, and 5 and go directly to section 6. In section 7 we show how to use our techniques to prove that 2-local adiabatic computation is equivalent to standard quantum computation. Some open questions are mentioned in section 8.

2. Preliminaries. QMA is naturally defined as a class of promise problems: A promise problem L is a pair (L_{yes}, L_{no}) of disjoint sets of strings corresponding to YES and NO instances of the problem. The problem is to determine, given a string $x \in L_{yes} \cup L_{no}$, whether $x \in L_{yes}$ or $x \in L_{no}$. Let \mathcal{B} be the Hilbert space of a qubit.

DEFINITION 2.1 (QMA). Fix $\varepsilon = \varepsilon(|x|)$ such that $\varepsilon = 2^{-\Omega(|x|)}$. Then, a promise problem L is in QMA if there exists a quantum polynomial time verifier V and a polynomial p such that

³Interestingly, their proof uses ideas from the proof of QMA-completeness of the LOCAL HAMILTONIAN problem.

⁴The main result of [3] is that 2-local adiabatic computation on *six-dimensional particles* is equivalent to standard quantum computation. This result is incomparable to ours since the particles in [3] are set on a two-dimensional grid and all 2-local interactions are between closest neighbors.

- $\forall x \in L_{yes} \exists |\xi\rangle \in \mathcal{B}^{\otimes p(|x|)}$ such that $\Pr(V(|x\rangle, |\xi\rangle) = 1) \geq 1 - \varepsilon$,
- $\forall x \in L_{no} \forall |\xi\rangle \in \mathcal{B}^{\otimes p(|x|)}$ such that $\Pr(V(|x\rangle, |\xi\rangle) = 1) \leq \varepsilon$,

where $\Pr(V(|x\rangle, |\xi\rangle) = 1)$ denotes the probability that V outputs 1 given $|x\rangle$ and $|\xi\rangle$.

We note that in the original definition ε was defined to be $2^{-\Omega(|x|)} \leq \varepsilon \leq 1/3$. By using amplification methods, it was shown in [12] that for any choice of ε in this range the resulting classes are equivalent. Hence our definition is equivalent to the original one. In a related result, Marriott and Watrous [14] showed that exponentially small ε can be achieved without amplification with a polynomial overhead in the verifier’s computation.

A natural choice for the quantum analogue of SAT is the LOCAL HAMILTONIAN problem. As we will see later, this problem is indeed a complete problem for QMA.

DEFINITION 2.2. We say that an operator $H : \mathcal{B}^{\otimes n} \rightarrow \mathcal{B}^{\otimes n}$ on n qubits is a k -local Hamiltonian if H is expressible as $H = \sum_{j=1}^r H_j$ where each term is a Hermitian operator acting on at most k qubits.

DEFINITION 2.3. The (promise) problem k -LOCAL HAMILTONIAN is defined as follows. We are given a k -local Hamiltonian on n qubits $H = \sum_{j=1}^r H_j$ with $r = \text{poly}(n)$. Each H_j has a bounded operator norm $\|H_j\| \leq \text{poly}(n)$, and its entries are specified by $\text{poly}(n)$ bits. In addition, we are given two constants a and b with $a < b$. In YES instances, the smallest eigenvalue of H is at most a . In NO instances, it is larger than b . We should decide which one is the case.

We will frequently refer to the lowest eigenvalue of some Hamiltonian H .

DEFINITION 2.4. Let $\lambda(H)$ denote the lowest eigenvalue of the Hamiltonian H . Another important notion that will be used in this paper is that of a restriction of a Hamiltonian.

DEFINITION 2.5. Let H be a Hamiltonian and let Π be a projection on some subspace \mathcal{S} . Then we say that the Hamiltonian $\Pi H \Pi$ on \mathcal{S} is the restriction of H to \mathcal{S} . We denote this restriction by $H|_{\mathcal{S}}$.

3. Projection lemma. Our main technical tool is the *projection lemma*. This lemma (in a slightly different form) was already used in [11] and [3] but not as extensively as it is used in this paper (in fact, we apply it four times in the first proof of our main theorem). The lemma allows us to successively *cut out* parts of the Hilbert space by giving them a large *penalty*. More precisely, assume we work in some Hilbert space \mathcal{H} and let H_1 be some Hamiltonian. For some subspace $\mathcal{S} \subseteq \mathcal{H}$, let H_2 be a Hamiltonian with the property that \mathcal{S} is an eigenspace of eigenvalue 0 and \mathcal{S}^\perp has eigenvalues at least J for some large $J \gg \|H_1\|$. In other words, H_2 gives a very high penalty to states in \mathcal{S}^\perp . Now consider the Hamiltonian $H = H_1 + H_2$. The projection lemma says that $\lambda(H)$, the lowest eigenvalue of H , is very close to $\lambda(H_1|_{\mathcal{S}})$, the lowest eigenvalue of the restriction of H_1 to \mathcal{S} . The intuitive reason for this is the following. By adding H_2 we give a very high penalty to any vector that has even a small projection in the \mathcal{S}^\perp direction. Hence, all eigenvectors with low eigenvalue (and in particular the one corresponding to $\lambda(H)$) have to lie very close to \mathcal{S} . From this it follows that these eigenvectors correspond to the eigenvectors of $H_1|_{\mathcal{S}}$.

The strength of this lemma comes from the following fact. Even though H_1 and H_2 are local Hamiltonians, $H_1|_{\mathcal{S}}$ is not necessarily so. In other words, the projection lemma allows us to approximate a nonlocal Hamiltonian by a local Hamiltonian.

LEMMA 3.1. Let $H = H_1 + H_2$ be the sum of two Hamiltonians operating on some Hilbert space $\mathcal{H} = \mathcal{S} + \mathcal{S}^\perp$. The Hamiltonian H_2 is such that \mathcal{S} is a zero eigenspace

and the eigenvectors in \mathcal{S}^\perp have eigenvalue at least $J > 2\|H_1\|$. Then,

$$\lambda(H_1|_{\mathcal{S}}) - \frac{\|H_1\|^2}{J - 2\|H_1\|} \leq \lambda(H) \leq \lambda(H_1|_{\mathcal{S}}).$$

Notice that with, say, $J \geq 8\|H_1\|^2 + 2\|H_1\| = \text{poly}(\|H_1\|)$ we have $\lambda(H_1|_{\mathcal{S}}) - 1/8 \leq \lambda(H) \leq \lambda(H_1|_{\mathcal{S}})$.

Proof. First, we show that $\lambda(H) \leq \lambda(H_1|_{\mathcal{S}})$. Let $|\eta\rangle \in \mathcal{S}$ be the eigenvector of $H_1|_{\mathcal{S}}$ corresponding to $\lambda(H_1|_{\mathcal{S}})$. Using $H_2|\eta\rangle = 0$,

$$\langle \eta | H | \eta \rangle = \langle \eta | H_1 | \eta \rangle + \langle \eta | H_2 | \eta \rangle = \lambda(H_1|_{\mathcal{S}})$$

and hence H must have an eigenvector of eigenvalue at most $\lambda(H_1|_{\mathcal{S}})$.

We now show the lower bound on $\lambda(H)$. We can write any unit vector $|v\rangle \in \mathcal{H}$ as $|v\rangle = \alpha_1|v_1\rangle + \alpha_2|v_2\rangle$ where $|v_1\rangle \in \mathcal{S}$ and $|v_2\rangle \in \mathcal{S}^\perp$ are two unit vectors, $\alpha_1, \alpha_2 \in \mathbb{R}$, $\alpha_1, \alpha_2 \geq 0$ and $\alpha_1^2 + \alpha_2^2 = 1$. Let $K = \|H_1\|$. Then we have

$$\begin{aligned} \langle v | H | v \rangle &\geq \langle v | H_1 | v \rangle + J\alpha_2^2 \\ &= (1 - \alpha_2^2)\langle v_1 | H_1 | v_1 \rangle + 2\alpha_1\alpha_2\text{Re}\langle v_1 | H_1 | v_2 \rangle + \alpha_2^2\langle v_2 | H_1 | v_2 \rangle + J\alpha_2^2 \\ &\geq \langle v_1 | H_1 | v_1 \rangle - K\alpha_2^2 - 2K\alpha_2 - K\alpha_2^2 + J\alpha_2^2 \\ &= \langle v_1 | H_1 | v_1 \rangle + (J - 2K)\alpha_2^2 - 2K\alpha_2 \\ &\geq \lambda(H_1|_{\mathcal{S}}) + (J - 2K)\alpha_2^2 - 2K\alpha_2, \end{aligned}$$

where we used $\alpha_1^2 = 1 - \alpha_2^2$ and $\alpha_1 \leq 1$. Since $(J - 2K)\alpha_2^2 - 2K\alpha_2$ is minimized for $\alpha_2 = K/(J - 2K)$, we have

$$\langle v | H | v \rangle \geq \lambda(H_1|_{\mathcal{S}}) - \frac{K^2}{J - 2K}. \quad \square$$

4. Kitaev’s construction. In this section we reprove Kitaev’s result that $O(\log n)$ -LOCAL HAMILTONIAN is QMA-complete. The difference between our version of the proof and the original one in [12] is that we do not use the authors’ geometrical lemma to obtain the result, but rather apply our Lemma 3.1. This paves the way to the later proof that 2-LOCAL HAMILTONIAN is QMA-complete.

As mentioned before, the proof that $O(\log n)$ -LOCAL HAMILTONIAN is in QMA appears in [12]. Hence, our goal is to show that any problem in QMA can be reduced to $O(\log n)$ -LOCAL HAMILTONIAN. Let $V_x = V(|x\rangle, \cdot) = U_T \cdots U_1$ be a quantum verifier circuit of size $T = \text{poly}(|x|)$ operating on $N = \text{poly}(|x|)$ qubits.⁵ Here and in what follows we assume without loss of generality that each U_i is either a one-qubit gate or a two-qubit gate. We further assume that $T \geq N$ and that initially, the first $m = p(|x|)$ qubits contain the proof and the remaining ancillary $N - m$ qubits are zero (see Definition 2.1). Finally, we assume that the output of the circuit is written into the first qubit (i.e., it is $|1\rangle$ if the circuit accepts). See Figure 4.1.

⁵For ease of notation we hardwire the dependence on the input x into the circuit.

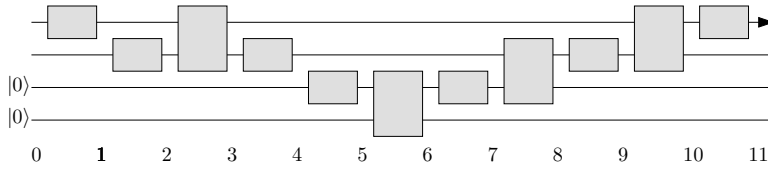


FIG. 4.1. A circuit with $T = 11$, $N = 4$, and $m = 2$.

The constructed Hamiltonian H operates on a space of $n = N + \log(T + 1)$ qubits. The first N qubits represent the computation and the last $\log(T + 1)$ qubits represent the possible values $0, \dots, T$ for the clock:

$$H = H_{out} + J_{in}H_{in} + J_{prop}H_{prop}.$$

The coefficients J_{in} and J_{prop} will be chosen later to be some large polynomials in N . The terms are given by

$$(4.1) \quad \begin{aligned} H_{in} &= \sum_{i=m+1}^N |1\rangle\langle 1|_i \otimes |0\rangle\langle 0|, & H_{out} &= (T + 1)|0\rangle\langle 0|_1 \otimes |T\rangle\langle T|, \\ H_{prop} &= \sum_{t=1}^T H_{prop,t}, \end{aligned}$$

and

$$(4.2) \quad H_{prop,t} = \frac{1}{2} \left(I \otimes |t\rangle\langle t| + I \otimes |t-1\rangle\langle t-1| - U_t \otimes |t\rangle\langle t-1| - U_t^\dagger \otimes |t-1\rangle\langle t| \right)$$

for $1 \leq t \leq T$, where $|\alpha\rangle\langle \alpha|_i$ denotes the projection on the subspace in which the i th qubit is $|\alpha\rangle$. It is understood that the first part of each tensor product acts on the space of the N computation qubits and the second part acts on the clock qubits. U_t and U_t^\dagger in $H_{prop,t}$ act on the same computational qubits as U_t does when it is employed in the verifier’s circuit V_x . Intuitively, each Hamiltonian “checks” a certain property by increasing the eigenvalue if the property does not hold: The Hamiltonian H_{in} checks that the input of the circuit is correct (i.e., none of the last $N - m$ computation qubits is 1), H_{out} checks that the output bit indicates acceptance, and H_{prop} checks that the propagation is according to the circuit. Notice that these Hamiltonians are $O(\log n)$ -local since there are $\log(T + 1) = O(\log n)$ clock qubits.

To show that a problem in QMA reduces to the $O(\log n)$ -LOCAL HAMILTONIAN problem with H chosen as above, we prove the following lemma.

LEMMA 4.1. *If the circuit V_x accepts with probability more than $1 - \varepsilon$ on some input $|\xi, 0\rangle$, then the Hamiltonian H has an eigenvalue smaller than ε . If the circuit V_x accepts with probability less than ε on all inputs $|\xi, 0\rangle$, then all eigenvalues of H are larger than $\frac{3}{4} - \varepsilon$.*

Proof. Assume the circuit V_x accepts with probability more than $1 - \varepsilon$ on some $|\xi, 0\rangle$. Define

$$|\eta\rangle = \frac{1}{\sqrt{T + 1}} \sum_{t=0}^T U_t \cdots U_1 |\xi, 0\rangle \otimes |t\rangle.$$

It can be seen that $\langle \eta | H_{prop} | \eta \rangle = \langle \eta | H_{in} | \eta \rangle = 0$ and that $\langle \eta | H_{out} | \eta \rangle < \varepsilon$. Hence, the smallest eigenvalue of H is less than ε . It remains to prove the second part of

the lemma. So now assume the circuit V_x accepts with probability less than ε on all inputs $|\xi, 0\rangle$.

Let \mathcal{S}_{prop} be the groundspace of the Hamiltonian H_{prop} . It is easy to see that \mathcal{S}_{prop} is a 2^N -dimensional space whose basis is given by the states

$$(4.3) \quad |\eta_i\rangle = \frac{1}{\sqrt{T+1}} \sum_{t=0}^T U_t \cdots U_1 |i\rangle \otimes |t\rangle,$$

where $i \in \{0, \dots, 2^N - 1\}$ and $|i\rangle$ represents the i th vector in the computational basis on the N computation qubits. These states have eigenvalue 0. The states in \mathcal{S}_{prop} represent the correct propagation from an initial state on the N computation qubits according to the verifier's circuit V_x .

We would like to apply Lemma 3.1 with the space \mathcal{S}_{prop} . For that, we need to establish that $J_{prop}H_{prop}$ gives a sufficiently large ($\text{poly}(N)$) penalty to states in \mathcal{S}_{prop}^\perp . In other words, the smallest nonzero eigenvalue of H_{prop} has to be lower bounded by some inverse polynomial in N . This has been shown in [12], but we wish to briefly recall it here, as it will apply in several instances throughout this paper.

CLAIM 4.2 ([12]). *The smallest nonzero eigenvalue of H_{prop} is at least c/T^2 for some constant $c > 0$.*

Proof. We first apply the change of basis

$$W = \sum_{t=0}^T U_t \cdots U_1 \otimes |t\rangle\langle t|,$$

which transforms H_{prop} to

$$W^\dagger H_{prop} W = \sum_{t=1}^T I \otimes \frac{1}{2} (|t\rangle\langle t| + |t-1\rangle\langle t-1| - |t\rangle\langle t-1| - |t-1\rangle\langle t|).$$

The eigenspectrum of H_{prop} is unchanged by this transformation. The resulting Hamiltonian is block-diagonal with 2^N blocks of size $T + 1$:

$$(4.4) \quad W^\dagger H_{prop} W = I \otimes \begin{pmatrix} \frac{1}{2} & -\frac{1}{2} & 0 & \cdots & 0 \\ -\frac{1}{2} & 1 & -\frac{1}{2} & 0 & \ddots & \vdots \\ 0 & -\frac{1}{2} & 1 & -\frac{1}{2} & 0 & \ddots & \vdots \\ & \ddots & \ddots & \ddots & \ddots & \ddots & \\ \vdots & & 0 & -\frac{1}{2} & 1 & -\frac{1}{2} & 0 \\ 0 & \cdots & & 0 & -\frac{1}{2} & 1 & -\frac{1}{2} \\ & & & & & & \frac{1}{2} \end{pmatrix}.$$

Using standard techniques, one can show that the smallest nonzero eigenvalue of each $(T + 1) \times (T + 1)$ block matrix is bounded from below by c/T^2 for some constant $c > 0$. \square

Hence any eigenvector of $J_{prop}H_{prop}$ orthogonal to \mathcal{S}_{prop} has eigenvalue at least $J = cJ_{prop}/T^2$. Let us apply Lemma 3.1 with

$$H_1 = H_{out} + J_{in}H_{in}, \quad H_2 = J_{prop}H_{prop}.$$

Note that $\|H_1\| \leq \|H_{out}\| + J_{in}\|H_{in}\| \leq T + 1 + J_{in}N \leq \text{poly}(N)$ since H_{in} and H_{out} are sums of orthogonal projectors and $J_{in} = \text{poly}(N)$. Lemma 3.1 implies that we can choose $J_{prop} = JT^2/c = \text{poly}(N)$, such that $\lambda(H)$ is lower bounded by $\lambda(H_1|_{\mathcal{S}_{prop}}) - \frac{1}{8}$. With this in mind, let us now consider the Hamiltonian $H_1|_{\mathcal{S}_{prop}}$ on \mathcal{S}_{prop} .

Let $\mathcal{S}_{in} \subset \mathcal{S}_{prop}$ be the groundspace of $H_{in}|_{\mathcal{S}_{prop}}$. Then \mathcal{S}_{in} is a 2^m -dimensional space whose basis is given by states as in (4.3) with $|i\rangle = |j, 0\rangle$, where $|j\rangle$ is a computational basis state on the first m computation qubits. We apply Lemma 3.1 again inside \mathcal{S}_{prop} with

$$H_1 = H_{out}|_{\mathcal{S}_{prop}}, \quad H_2 = J_{in}H_{in}|_{\mathcal{S}_{prop}}.$$

This time, $\|H_1\| \leq \|H_{out}\| = T + 1 = \text{poly}(N)$. Any eigenvector of H_2 orthogonal to \mathcal{S}_{in} inside \mathcal{S}_{prop} has eigenvalue at least $J_{in}/(T + 1)$. Hence, there is a $J_{in} = \text{poly}(N)$ such that $\lambda(H_1 + H_2)$ is lower bounded by $\lambda(H_{out}|_{\mathcal{S}_{in}}) - \frac{1}{8}$.

Since the circuit V_x accepts with probability less than ε on all inputs $|\xi, 0\rangle$, we have that all eigenvalues of $H_{out}|_{\mathcal{S}_{in}}$ are larger than $1 - \varepsilon$. Hence the smallest eigenvalue of H is larger than $1 - \varepsilon - \frac{2}{8} = \frac{3}{4} - \varepsilon$, proving the second part of the lemma. \square

5. The 2-local construction.

Previous constructions. Let us give an informal description of ideas used in previous improvements on Kitaev’s construction; these ideas will also appear in our proof. The first idea is to represent the clock register in *unary notation*. Then, the clock register consists of T qubits, and time step $t \in \{0, \dots, T\}$ is represented by $|1^t 0^{T-t}\rangle$. The crucial observation is that clock terms that used to involve $\log(T + 1)$ qubits can now be replaced by 3-local terms that are essentially equivalent. For example, a term like $|t-1\rangle\langle t|$ can be replaced by the term $|100\rangle\langle 110|_{t-1,t,t+1}$. Since the gates U_t involve at most two qubits, we obtain a 5-local Hamiltonian. This is essentially the way 5-LOCAL HAMILTONIAN was shown to be QMA-complete in [12]. The only minor complication is that we need to get rid of illegal clock states (i.e., ones that are not a unary representation). This is done by the addition of a (2-local) Hamiltonian H_{clock} that penalizes a clock state whenever 1 appears after 0.

This result was further improved to 3-LOCAL HAMILTONIAN in [11]. The main idea there is to replace a 3-local clock term like $|100\rangle\langle 110|_{t-1,t,t+1}$ by the 1-local term $|0\rangle\langle 1|_t$. These one-qubit terms are no longer equivalent to the original clock terms. Indeed, it can be seen that they have unwanted transitions into illegal clock states. The main idea in [11] was that by giving a large penalty to illegal clock states (i.e., by multiplying H_{clock} by some large number) and applying the projection lemma, we can essentially project these one-qubit terms to the subspace of legal clock states. Inside this subspace, these terms become the required clock terms.

The 2-local construction. Most of the terms that appear in the construction of [11] are already 2-local. The only 3-local terms are terms as in (4.2) that correspond to two-qubit gates (those corresponding to one-qubit gates are already 2-local). Hence, in order to prove our main theorem, it is enough to find a 2-local Hamiltonian that checks for the correct propagation of two-qubit gates. This seems difficult because the Hamiltonian must somehow couple two computation qubits to a clock qubit. We circumvent this problem in the following manner. First, we isolate from the propagation Hamiltonian those terms that correspond to one-qubit gates and we multiply these terms by some large factor. Using the projection lemma, we can project the remaining Hamiltonians into a space where the one-qubit-gate propagation is correct. In other words, at this stage we can assume that our space is spanned by states that correspond to legal propagation according to the one-qubit gates. This allows us to couple clock qubits *instead* of computation qubits. To see this, consider the circuit in

Figure 5.1 at time t and at time $t + 2$. A Z gate flips the phase of a qubit if its state is $|1\rangle$ and leaves it unchanged otherwise. Hence, the phase difference between time t and time $t + 2$ corresponds to the parity of the two qubits. This phase difference can be detected by a 2-local term such as $|00\rangle\langle 11|_{t+1,t+2}$. The crucial point here is that by using a term involving only two clock qubits, we are able to check the state of two computation qubits (in this case, their parity) at a certain time. This is the main idea in our proof.

We now present the proof of the main theorem in detail. We start by making some further assumptions on the circuit V_x , all without loss of generality. First, we assume that in addition to one-qubit gates, the circuit contains only the controlled phase gate, C_ϕ . This two-qubit gate is diagonal in the computational basis and flips the sign of the state $|11\rangle$,

$$C_\phi = C_\phi^\dagger = |00\rangle\langle 00| + |01\rangle\langle 01| + |10\rangle\langle 10| - |11\rangle\langle 11|.$$

It is known [5, 15] that quantum circuits consisting of one-qubit gates and C_ϕ gates are universal⁶ and can simulate any other quantum circuit with only polynomial overhead. Second, we assume that each C_ϕ gate is both preceded and followed by two Z gates, one on each qubit, as in Figure 5.1. The Z gate is defined by $|0\rangle\langle 0| - |1\rangle\langle 1|$; i.e., it is a diagonal one-qubit gate that flips the sign of $|1\rangle$. Since both the Z gate and the C_ϕ gate are diagonal, they commute and the effect of the Z -gates cancels out. This assumption makes the circuit at most five times bigger. Finally, we assume that the C_ϕ gates are applied at regular intervals. In other words, if T_2 is the number of C_ϕ gates and L is the interval length, then a C_ϕ gate is applied at steps $L, 2L, \dots, T_2L$. Before the first C_ϕ gate, after the last C_ϕ gate, and between any two consecutive C_ϕ gates we have $L - 1$ one-qubit gates. This makes the total number of gates in the resulting circuit $T = (T_2 + 1)L - 1$.

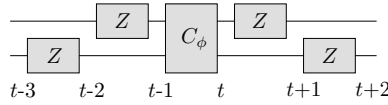


FIG. 5.1. A modified C_ϕ gate applied at step t .

We construct a Hamiltonian H that operates on a space of $N + T$ qubits. The first N qubits represent the computation and the last T qubits represent the clock. We think of the clock as represented in unary,

$$(5.1) \quad |\hat{t}\rangle \stackrel{def}{=} |\underbrace{1\dots 1}_t \underbrace{0\dots 0}_{T-t}\rangle.$$

Let T_1 be the time steps in which a one-qubit gate is applied. Namely, $T_1 = \{1, \dots, T\} \setminus \{L, 2L, \dots, T_2L\}$. Then

$$H = H_{out} + J_{in}H_{in} + J_2H_{prop2} + J_1H_{prop1} + J_{clock}H_{clock},$$

⁶The original universal gate set in [5] consists of one-qubit gates and CNOT gates. It is, however, easy to see that a CNOT gate can be obtained from a C_ϕ gate by conjugating the second qubit with Hadamard gates (see [15]).

where

$$H_{in} = \sum_{i=m+1}^N |1\rangle\langle 1|_i \otimes |0\rangle\langle 0|_1, \quad H_{out} = (T+1)|0\rangle\langle 0|_1 \otimes |1\rangle\langle 1|_T,$$

$$H_{clock} = \sum_{1 \leq i < j \leq T} I \otimes |01\rangle\langle 01|_{ij}.$$

The terms H_{prop1} and H_{prop2} , which represent the correct propagation according to the one-qubit gates and two-qubit gates, respectively, are defined as

$$H_{prop1} = \sum_{t \in T_1} H_{prop,t}, \quad H_{prop2} = \sum_{l=1}^{T_2} (H_{qubit,lL} + H_{time,lL})$$

with

$$H_{prop,t} = \frac{1}{2} \left(I \otimes |10\rangle\langle 10|_{t,t+1} + I \otimes |10\rangle\langle 10|_{t-1,t} - U_t \otimes |1\rangle\langle 0|_t - U_t^\dagger \otimes |0\rangle\langle 1|_t \right)$$

for $t \in T_1 \cap \{2, \dots, T-1\}$ and

$$H_{prop,1} = \frac{1}{2} \left(I \otimes |10\rangle\langle 10|_{1,2} + I \otimes |0\rangle\langle 0|_1 - U_1 \otimes |1\rangle\langle 0|_1 - U_1^\dagger \otimes |0\rangle\langle 1|_1 \right),$$

$$H_{prop,T} = \frac{1}{2} \left(I \otimes |1\rangle\langle 1|_T + I \otimes |10\rangle\langle 10|_{T-1,T} - U_T \otimes |1\rangle\langle 0|_T - U_T^\dagger \otimes |0\rangle\langle 1|_T \right)$$

and, with f_t and s_t being the first and second qubits of the C_ϕ gate at time t ,

$$H_{qubit,t} = \frac{1}{2} \left(-2|0\rangle\langle 0|_{f_t} - 2|0\rangle\langle 0|_{s_t} + |1\rangle\langle 1|_{f_t} + |1\rangle\langle 1|_{s_t} \right) \otimes (|1\rangle\langle 0|_t + |0\rangle\langle 1|_t),$$

$$H_{time,t} = \frac{1}{8} I \otimes \left(|10\rangle\langle 10|_{t,t+1} + 6|10\rangle\langle 10|_{t+1,t+2} + |10\rangle\langle 10|_{t+2,t+3} \right. \\ \left. + 2|11\rangle\langle 00|_{t+1,t+2} + 2|00\rangle\langle 11|_{t+1,t+2} \right. \\ \left. + |1\rangle\langle 0|_{t+1} + |0\rangle\langle 1|_{t+1} + |1\rangle\langle 0|_{t+2} + |0\rangle\langle 1|_{t+2} \right. \\ \left. + |10\rangle\langle 10|_{t-3,t-2} + 6|10\rangle\langle 10|_{t-2,t-1} + |10\rangle\langle 10|_{t-1,t} \right. \\ \left. + 2|11\rangle\langle 00|_{t-2,t-1} + 2|00\rangle\langle 11|_{t-2,t-1} \right. \\ \left. + |1\rangle\langle 0|_{t-2} + |0\rangle\langle 1|_{t-2} + |1\rangle\langle 0|_{t-1} + |0\rangle\langle 1|_{t-1} \right).$$

At this point, these last two expressions might look strange. Let us say that later, when we consider their restriction to a smaller space, the reason for this definition should become clear. Note that all the above terms are at most 2-local. We will later choose $J_{in} \ll J_2 \ll J_1 \ll J_{clock} \leq \text{poly}(N)$. As in section 4, we have to prove the following lemma.

LEMMA 5.1. *Assume that the circuit V_x accepts with probability more than $1 - \varepsilon$ on some input $|\xi, 0\rangle$. Then H has an eigenvalue smaller than ε . If the circuit V_x accepts with probability less than ε on all inputs $|\xi, 0\rangle$, then all eigenvalues of H are larger than $\frac{1}{2} - \varepsilon$.*

Proof. If the circuit V_x accepts with probability more than $1 - \varepsilon$ on some input $|\xi, 0\rangle$, then the state

$$|\eta\rangle = \frac{1}{\sqrt{T+1}} \sum_{t=0}^T U_t \cdots U_1 |\xi, 0\rangle \otimes |\hat{t}\rangle$$

satisfies $\langle \eta | H | \eta \rangle \leq \varepsilon$. In order to see this, one can check that

$$\langle \eta | H_{clock} | \eta \rangle = \langle \eta | H_{prop1} | \eta \rangle = \langle \eta | H_{prop2} | \eta \rangle = \langle \eta | H_{in} | \eta \rangle = 0$$

and $\langle \eta | H_{out} | \eta \rangle \leq \varepsilon$. However, verifying that $\langle \eta | H_{prop2} | \eta \rangle = 0$ can be quite tedious. Later in the proof, we will mention an easier way to see this.

In the following, we will show that if the circuit V_x accepts with probability less than ε on all inputs $|\xi, 0\rangle$, then all eigenvalues of H are larger than $\frac{1}{2} - \varepsilon$. The proof of this is based on four applications of Lemma 3.1. Schematically, we proceed as follows:

$$\mathcal{H} \supset \mathcal{S}_{legal} \supset \mathcal{S}_{prop1} \supset \mathcal{S}_{prop} \supset \mathcal{S}_{in},$$

where \mathcal{S}_{legal} corresponds to states with *legal* clock states written in unary, and \mathcal{S}_{prop1} is spanned by states in the legal clock space whose propagation at time steps corresponding to *one-qubit* gates (that is, in T_1) is correct. Finally, \mathcal{S}_{prop} and \mathcal{S}_{in} are defined in almost the same way as in section 4. These spaces will be described in more detail later.

Norms. Note that all relevant norms, as needed in Lemma 3.1, are polynomial in N . Indeed, we have $\|H_{out}\| = T + 1$ and $\|H_{in}\| \leq N$ as in section 4, $\|H_{prop1}\| \leq \sum_{t \in T_1} \|H_{prop,t}\| \leq 2T$ (each term in H_{prop1} has norm at most 2), and $\|H_{prop2}\| \leq \sum_{t=1}^{T_2} (\|H_{qubit,t}\| + \|H_{time,t}\|) \leq O(T_2) \leq O(T)$.

1. *Restriction to legal clock states in \mathcal{S}_{legal} .* Let \mathcal{S}_{legal} be the $(T+1)2^N$ -dimensional space spanned by states with a legal unary representation on the T clock qubits, i.e., by states of the form $|\tilde{\xi}\rangle \otimes |\hat{t}\rangle$ with $|\hat{t}\rangle$ as in (5.1). In this first stage we apply Lemma 3.1 with

$$H_1 = H_{out} + J_{in}H_{in} + J_2H_{prop2} + J_1H_{prop1}, \quad H_2 = J_{clock}H_{clock}.$$

Notice that \mathcal{S}_{legal} is an eigenspace of H_2 of eigenvalue 0 and that states orthogonal to \mathcal{S}_{legal} have eigenvalue at least J_{clock} . Lemma 3.1 implies that we can choose $J_{clock} = \text{poly}(\|H_1\|) = \text{poly}(N)$ such that $\lambda(H)$ can be lower bounded by $\lambda(H_1|_{\mathcal{S}_{legal}}) - \frac{1}{8}$. Hence, in the remainder of the proof, it is enough to study $H_1|_{\mathcal{S}_{legal}}$ inside the space \mathcal{S}_{legal} . This can be written as

$$H_{out}|_{\mathcal{S}_{legal}} + J_{in}H_{in}|_{\mathcal{S}_{legal}} + J_2H_{prop2}|_{\mathcal{S}_{legal}} + J_1H_{prop1}|_{\mathcal{S}_{legal}}$$

with

$$\begin{aligned} H_{in}|_{\mathcal{S}_{legal}} &= \sum_{i=m+1}^N |1\rangle\langle 1|_i \otimes |\hat{0}\rangle\langle \hat{0}|, & H_{out}|_{\mathcal{S}_{legal}} &= (T+1)|0\rangle\langle 0|_1 \otimes |\hat{T}\rangle\langle \hat{T}|, \\ H_{prop,t}|_{\mathcal{S}_{legal}} &= \frac{1}{2} \left(I \otimes |\hat{t}\rangle\langle \hat{t}| + I \otimes |\hat{t-1}\rangle\langle \hat{t-1}| - U_t \otimes |\hat{t}\rangle\langle \hat{t-1}| - U_t^\dagger \otimes |\hat{t-1}\rangle\langle \hat{t}| \right), \\ H_{qubit,t}|_{\mathcal{S}_{legal}} &= \frac{1}{2} \left(-2|0\rangle\langle 0|_{f_t} - 2|0\rangle\langle 0|_{s_t} + |1\rangle\langle 1|_{f_t} + |1\rangle\langle 1|_{s_t} \right) \otimes \left(|\hat{t}\rangle\langle \hat{t-1}| + |\hat{t-1}\rangle\langle \hat{t}| \right), \\ H_{time,t}|_{\mathcal{S}_{legal}} &= \frac{1}{8} I \otimes \left(|\hat{t}\rangle\langle \hat{t}| + 6|\hat{t+1}\rangle\langle \hat{t+1}| + |\hat{t+2}\rangle\langle \hat{t+2}| \right. \\ &\quad + 2|\hat{t+2}\rangle\langle \hat{t}| + 2|\hat{t}\rangle\langle \hat{t+2}| + |\hat{t+1}\rangle\langle \hat{t}| + |\hat{t}\rangle\langle \hat{t+1}| + |\hat{t+2}\rangle\langle \hat{t+1}| + |\hat{t+1}\rangle\langle \hat{t+2}| \\ &\quad + |\hat{t-3}\rangle\langle \hat{t-3}| + 6|\hat{t-2}\rangle\langle \hat{t-2}| + |\hat{t-1}\rangle\langle \hat{t-1}| \\ &\quad \left. + 2|\hat{t-1}\rangle\langle \hat{t-3}| + 2|\hat{t-3}\rangle\langle \hat{t-1}| + |\hat{t-2}\rangle\langle \hat{t-3}| + |\hat{t-3}\rangle\langle \hat{t-2}| + |\hat{t-1}\rangle\langle \hat{t-2}| + |\hat{t-2}\rangle\langle \hat{t-1}| \right). \end{aligned}$$

The above was obtained by noting that the projection of a term like, say, $|10\rangle\langle 10|_{t,t+1}$ on \mathcal{S}_{legal} is exactly $|\hat{t}\rangle\langle \hat{t}|$. Similarly, the projection of the term $|1\rangle\langle 0|_{t+1}$ is $|\widehat{t+1}\rangle\langle \hat{t}|$.⁷ By rearranging terms, $H_{time,t}|_{\mathcal{S}_{legal}}$ can be written as a sum of projectors:

$$\begin{aligned}
 & \frac{1}{8}I \otimes \left\{ 2 \left(|\hat{t}\rangle + |\widehat{t+1}\rangle \right) \left(\langle \hat{t}| + \langle \widehat{t+1}| \right) + 2 \left(|\widehat{t+1}\rangle + |\widehat{t+2}\rangle \right) \left(\langle \widehat{t+1}| + \langle \widehat{t+2}| \right) \right. \\
 & \quad + \left(|\hat{t}\rangle - |\widehat{t+1}\rangle \right) \left(\langle \hat{t}| - \langle \widehat{t+1}| \right) + \left(|\widehat{t+1}\rangle - |\widehat{t+2}\rangle \right) \left(\langle \widehat{t+1}| - \langle \widehat{t+2}| \right) \\
 & \quad - 2 \left(|\hat{t}\rangle - |\widehat{t+2}\rangle \right) \left(\langle \hat{t}| - \langle \widehat{t+2}| \right) \\
 & \quad + 2 \left(|\widehat{t-3}\rangle + |\widehat{t-2}\rangle \right) \left(\langle \widehat{t-3}| + \langle \widehat{t-2}| \right) + 2 \left(|\widehat{t-2}\rangle + |\widehat{t-1}\rangle \right) \left(\langle \widehat{t-2}| + \langle \widehat{t-1}| \right) \\
 & \quad + \left(|\widehat{t-3}\rangle - |\widehat{t-2}\rangle \right) \left(\langle \widehat{t-3}| - \langle \widehat{t-2}| \right) + \left(|\widehat{t-2}\rangle - |\widehat{t-1}\rangle \right) \left(\langle \widehat{t-2}| - \langle \widehat{t-1}| \right) \\
 & \quad \left. - 2 \left(|\widehat{t-3}\rangle - |\widehat{t-1}\rangle \right) \left(\langle \widehat{t-3}| - \langle \widehat{t-1}| \right) \right\}.
 \end{aligned}
 \tag{5.2}$$

Notice that the above expression is symmetric around $t - \frac{1}{2}$ (i.e., switching $t - 1$ with t , $t - 2$ with $t + 1$, and $t - 3$ with $t + 2$ does not change the expression). Let us also mention that the fact that we have terms like $|\hat{t}\rangle - |\widehat{t+2}\rangle$ is crucial in our proof. They allow us to compare the state at time t to the state at time $t + 2$.

2. *Restriction to \mathcal{S}_{prop1} .* We now apply Lemma 3.1 inside \mathcal{S}_{legal} with

$$H_1 = (H_{out} + J_{in}H_{in} + J_2H_{prop2})|_{\mathcal{S}_{legal}}, \quad H_2 = J_1H_{prop1}|_{\mathcal{S}_{legal}}.$$

Let \mathcal{S}_{prop1} be the $2^N(T_2 + 1)$ -dimensional space given by all states that represent correct propagation on all one-qubit gates. More precisely, let

$$|\eta_{l,i}\rangle \stackrel{def}{=} \frac{1}{\sqrt{L}} \sum_{t=lL}^{(l+1)L-1} U_t \cdots U_1 |i\rangle \otimes |\hat{t}\rangle,
 \tag{5.3}$$

where $l \in \{0, \dots, T_2\}$, $i \in \{0, \dots, 2^N - 1\}$, and $|i\rangle$ represents the i th vector in the computational basis. Then these states form a basis of \mathcal{S}_{prop1} . It is easy to see that each $|\eta_{l,i}\rangle$ is an eigenvector of H_{prop1} of eigenvalue 0. Hence, \mathcal{S}_{prop1} is an eigenspace of eigenvalue 0 of $H_{prop1}|_{\mathcal{S}_{legal}}$. Furthermore, $H_{prop1}|_{\mathcal{S}_{legal}}$ decomposes into $T_2 + 1$ invariant blocks, with the l th block spanned by states of the form $U_t \cdots U_1 |i\rangle \otimes |\hat{t}\rangle$ for $t = lL, \dots, (l + 1)L - 1$. Inside such a block $H_{prop1}|_{\mathcal{S}_{legal}}$ corresponds exactly to H_{prop} of section 4, equations (4.1), (4.2). By Claim 4.2, its nonzero eigenvalues are at least $c/L^2 \geq c/T^2$ for some constant $c > 0$ and hence the smallest nonzero eigenvalue of $H_{prop1}|_{\mathcal{S}_{legal}}$ is also at least c/T^2 . Therefore, all eigenvectors of H_2 orthogonal to \mathcal{S}_{prop1} have eigenvalue at least $J = J_1c/T^2$ and Lemma 3.1 implies that for $J_1 \geq \text{poly}(N)$, $\lambda(H_1 + H_2)$ can be lower bounded by $\lambda(H_1|_{\mathcal{S}_{prop1}}) - \frac{1}{8}$.

Hence, in the remainder of the proof, it is enough to study

$$H_{out}|_{\mathcal{S}_{prop1}} + J_{in}H_{in}|_{\mathcal{S}_{prop1}} + J_2H_{prop2}|_{\mathcal{S}_{prop1}}.$$

Let us find $H_{prop2}|_{\mathcal{S}_{prop1}}$. Let $t = lL$ be the time at which the l th C_ϕ gate is applied and consider the projection of a state $|\eta_{l,i}\rangle$ onto the space spanned by the computation

⁷Notice that we do not have terms like $|1\rangle\langle 1|_t$; its projection on \mathcal{S}_{legal} is not $|\hat{t}\rangle\langle \hat{t}|$ but rather $|\hat{t}\rangle\langle \hat{t}| + \cdots + |\widehat{T}\rangle\langle \widehat{T}|$.

qubits and $|\widehat{t}\rangle, |\widehat{t+1}\rangle, |\widehat{t+2}\rangle$. Since at time $t + 1$ (resp., $t + 2$) a Z gate is applied to qubit f_t (resp., s_t), this projection is a linear combination of the following four states:

$$\begin{aligned} & |00\rangle_{f_t, s_t} |\xi_{00}\rangle \otimes \left(|\widehat{t}\rangle + |\widehat{t+1}\rangle + |\widehat{t+2}\rangle \right), \\ & |01\rangle_{f_t, s_t} |\xi_{01}\rangle \otimes \left(|\widehat{t}\rangle + |\widehat{t+1}\rangle - |\widehat{t+2}\rangle \right), \\ & |10\rangle_{f_t, s_t} |\xi_{10}\rangle \otimes \left(|\widehat{t}\rangle - |\widehat{t+1}\rangle - |\widehat{t+2}\rangle \right), \\ & |11\rangle_{f_t, s_t} |\xi_{11}\rangle \otimes \left(|\widehat{t}\rangle - |\widehat{t+1}\rangle + |\widehat{t+2}\rangle \right), \end{aligned}$$

where $|\xi_{b_1 b_2}\rangle$ is an arbitrary state on the remaining $N - 2$ computation qubits. This implies that the restriction to \mathcal{S}_{prop1} of the projector on, say, $|\widehat{t}\rangle + |\widehat{t+1}\rangle$ from (5.2) is essentially the same as the restriction to \mathcal{S}_{prop1} of the projector on $|0\rangle_{f_t} |\widehat{t}\rangle$. More precisely, for all l_1, l_2, i_1, i_2 we have

$$\frac{1}{4} \langle \eta_{l_1, i_1} | \left(I \otimes (|\widehat{t}\rangle + |\widehat{t+1}\rangle) \right) (\langle \widehat{t} | + \langle \widehat{t+1} |) | \eta_{l_2, i_2} \rangle = \langle \eta_{l_1, i_1} | \left(|0\rangle \langle 0|_{f_t} \otimes |\widehat{t}\rangle \langle \widehat{t}| \right) | \eta_{l_2, i_2} \rangle.$$

Similarly, the term involving $|\widehat{t}\rangle - |\widehat{t+2}\rangle$ satisfies

$$\begin{aligned} & \frac{1}{4} \langle \eta_{l_1, i_1} | \left(I \otimes (|\widehat{t}\rangle - |\widehat{t+2}\rangle) \right) (\langle \widehat{t} | - \langle \widehat{t+2} |) | \eta_{l_2, i_2} \rangle \\ & = \langle \eta_{l_1, i_1} | \left((|01\rangle \langle 01|_{f_t, s_t} + |10\rangle \langle 10|_{f_t, s_t}) \otimes |\widehat{t}\rangle \langle \widehat{t}| \right) | \eta_{l_2, i_2} \rangle. \end{aligned}$$

Observe that the right-hand side involves two computation qubits and the clock register. Being able to obtain such a term from 2-local terms is a crucial ingredient in this proof.

Following a similar calculation, we see that from the terms involving $|\widehat{t-1}\rangle, |\widehat{t-2}\rangle, |\widehat{t-3}\rangle$ we obtain projectors involving $|\widehat{t-1}\rangle$. To summarize, instead of considering $H_{time, t} |_{\mathcal{S}_{prop1}}$ we can equivalently consider the restriction to \mathcal{S}_{prop1} of

$$\begin{aligned} & \frac{1}{2} \left(2|0\rangle \langle 0|_{f_t} + 2|0\rangle \langle 0|_{s_t} + |1\rangle \langle 1|_{f_t} + |1\rangle \langle 1|_{s_t} - 2|01\rangle \langle 01|_{f_t, s_t} - 2|10\rangle \langle 10|_{f_t, s_t} \right) \\ & \otimes \left(|\widehat{t-1}\rangle \langle \widehat{t-1}| + |\widehat{t}\rangle \langle \widehat{t}| \right). \end{aligned}$$

We now add the terms in $H_{qubit, t}$. A short calculation shows that the restriction to \mathcal{S}_{prop1} of $(H_{time, t} + H_{qubit, t})$ is the same as the restriction to \mathcal{S}_{prop1} of

$$\begin{aligned} & |00\rangle \langle 00|_{f_t, s_t} \otimes 2 \left(|\widehat{t-1}\rangle - |\widehat{t}\rangle \right) \left(\langle \widehat{t-1} | - \langle \widehat{t} | \right) + \\ & |01\rangle \langle 01|_{f_t, s_t} \otimes \frac{1}{2} \left(|\widehat{t-1}\rangle - |\widehat{t}\rangle \right) \left(\langle \widehat{t-1} | - \langle \widehat{t} | \right) + \\ & |10\rangle \langle 10|_{f_t, s_t} \otimes \frac{1}{2} \left(|\widehat{t-1}\rangle - |\widehat{t}\rangle \right) \left(\langle \widehat{t-1} | - \langle \widehat{t} | \right) + \\ & |11\rangle \langle 11|_{f_t, s_t} \otimes \left(|\widehat{t-1}\rangle + |\widehat{t}\rangle \right) \left(\langle \widehat{t-1} | + \langle \widehat{t} | \right). \end{aligned}$$

At this point, let us mention how one can show that for the state $|\eta\rangle$ described in the beginning of this proof, $\langle \eta | H_{prop2} | \eta \rangle = 0$. First, observe that $|\eta\rangle \in \mathcal{S}_{prop1}$ (its propagation is correct at all time steps). Next, since $|\eta\rangle$ has a C_ϕ propagation at time t , the above Hamiltonian shows that $\langle \eta | H_{prop2} | \eta \rangle = 0$.

Let us return now to the main proof. Recall that we wish to show a lower bound on the lowest eigenvalue of

$$(5.4) \quad H_{out}|_{\mathcal{S}_{prop1}} + J_{in}H_{in}|_{\mathcal{S}_{prop1}} + J_2H_{prop2}|_{\mathcal{S}_{prop1}}.$$

In the following, we show a lower bound on the lowest eigenvalue of the Hamiltonian

$$(5.5) \quad H_{out}|_{\mathcal{S}_{prop1}} + J_{in}H_{in}|_{\mathcal{S}_{prop1}} + J_2H'$$

on \mathcal{S}_{prop1} , where H' satisfies that $H' \leq H_{prop2}|_{\mathcal{S}_{prop1}}$, i.e., $H_{prop2}|_{\mathcal{S}_{prop1}} - H'$ is positive semidefinite. Hence, any lower bound on the lowest eigenvalue of the Hamiltonian in (5.5) implies the same lower bound on the lowest eigenvalue of the Hamiltonian in (5.4). We define H' as the sum over $t \in \{L, 2L, \dots, T_2L\}$ of the restriction to \mathcal{S}_{prop1} of

$$\begin{aligned} &|00\rangle\langle 00|_{f_t, s_t} \otimes \frac{1}{2} \left(|\widehat{t-1}\rangle - |\widehat{t}\rangle \right) \left(\langle \widehat{t-1}| - \langle \widehat{t}| \right) + \\ &|01\rangle\langle 01|_{f_t, s_t} \otimes \frac{1}{2} \left(|\widehat{t-1}\rangle - |\widehat{t}\rangle \right) \left(\langle \widehat{t-1}| - \langle \widehat{t}| \right) + \\ &|10\rangle\langle 10|_{f_t, s_t} \otimes \frac{1}{2} \left(|\widehat{t-1}\rangle - |\widehat{t}\rangle \right) \left(\langle \widehat{t-1}| - \langle \widehat{t}| \right) + \\ &|11\rangle\langle 11|_{f_t, s_t} \otimes \frac{1}{2} \left(|\widehat{t-1}\rangle + |\widehat{t}\rangle \right) \left(\langle \widehat{t-1}| + \langle \widehat{t}| \right). \end{aligned}$$

Equivalently, H' is the sum over $t \in \{L, 2L, \dots, T_2L\}$ of

$$\frac{1}{2} \left(I \otimes |\widehat{t}\rangle\langle \widehat{t}| + I \otimes |\widehat{t-1}\rangle\langle \widehat{t-1}| - C_\phi \otimes |\widehat{t}\rangle\langle \widehat{t-1}| - C_\phi^\dagger \otimes |\widehat{t-1}\rangle\langle \widehat{t}| \right) \Big|_{\mathcal{S}_{prop1}},$$

which resembles (4.2). Note that this term enforces correct propagation at time step $t = lL$. We claim that

$$(5.6) \quad H' = \frac{1}{2L} \sum_{i=0}^{2^N-1} \sum_{l=1}^{T_2} (|\eta_{l-1,i}\rangle - |\eta_{l,i}\rangle) (\langle \eta_{l-1,i}| - \langle \eta_{l,i}|).$$

The intuitive reason for this is the following. For any i , $|\eta_{l-1,i}\rangle + |\eta_{l,i}\rangle$ can be seen as a correct propagation at time $t = lL$. In other words, consider the projection of $|\eta_{l,i}\rangle$ on clock $|\widehat{t}\rangle$ and the projection of $|\eta_{l-1,i}\rangle$ on clock $|\widehat{t-1}\rangle$. Then the first state is exactly the second state after applying the l th C_ϕ gate. This means that inside \mathcal{S}_{prop1} , checking correct propagation from time $t-1$ to time t is equivalent to checking correct propagation from $|\eta_{l-1,i}\rangle$ to $|\eta_{l,i}\rangle$.

More precisely, fix some l and $t = lL$. Then, using (5.3), we get that for all l_1, l_2, i_1, i_2 such that either $l_1 \neq l$, $l_2 \neq l$, or $i_1 \neq i_2$,

$$\langle \eta_{l_1, i_1} | (I \otimes |\widehat{t}\rangle\langle \widehat{t}|) | \eta_{l_2, i_2} \rangle = 0.$$

Otherwise, $l_1 = l_2 = l$ and $i_1 = i_2 = i$ for some i and we have

$$\langle \eta_{l,i} | (I \otimes |\widehat{t}\rangle\langle \widehat{t}|) | \eta_{l,i} \rangle = \frac{1}{L}.$$

Hence we obtain

$$I \otimes |\widehat{t}\rangle\langle \widehat{t}| \Big|_{\mathcal{S}_{prop1}} = \frac{1}{L} \sum_{i=0}^{2^N-1} |\eta_{l,i}\rangle\langle \eta_{l,i}|$$

and similarly,

$$I \otimes |\widehat{t-1}\rangle\langle\widehat{t-1}|_{\mathcal{S}_{prop1}} = \frac{1}{L} \sum_{i=0}^{2^N-1} |\eta_{l-1,i}\rangle\langle\eta_{l-1,i}|.$$

For the off-diagonal terms we see that

$$\langle\eta_{l_1,i_1}| \left(C_\phi \otimes |\widehat{t}\rangle\langle\widehat{t-1}| \right) |\eta_{l_2,i_2}\rangle = 0$$

if $l_1 \neq l$ or $l_2 \neq l - 1$. If $l_1 = l$ and $l_2 = l - 1$, then using $C_\phi = U_{lL}$, we get

$$\langle\eta_{l,i_1}| \left(C_\phi \otimes |\widehat{t}\rangle\langle\widehat{t-1}| \right) |\eta_{l-1,i_2}\rangle = \frac{1}{L} \langle i_1 | (U_{lL} \cdots U_1)^\dagger C_\phi U_{lL-1} \cdots U_1 | i_2 \rangle = \frac{1}{L} \langle i_1 | i_2 \rangle,$$

which is 0 if $i_1 \neq i_2$ and $\frac{1}{L}$ otherwise. Hence $C_\phi \otimes |\widehat{t}\rangle\langle\widehat{t-1}|_{\mathcal{S}_{prop1}} = \frac{1}{L} \sum_{i=0}^{2^N-1} |\eta_{l,i}\rangle\langle\eta_{l-1,i}|$ and similarly for its Hermitian adjoint. This establishes (5.6).

3. *Restriction to \mathcal{S}_{prop} .* Let \mathcal{S}_{prop} be the 2^N -dimensional space whose basis is given by the states

$$|\eta_i\rangle = \frac{1}{\sqrt{T+1}} \sum_{t=0}^T U_t \cdots U_1 |i\rangle \otimes |\widehat{t}\rangle = \frac{1}{\sqrt{T_2+1}} \sum_{l=0}^{T_2} |\eta_{l,i}\rangle$$

for $i \in \{0, \dots, 2^N - 1\}$. Equation (5.6) shows that \mathcal{S}_{prop} is an eigenspace of H' of eigenvalue 0. Moreover, H' is block-diagonal with 2^N blocks of size $T_2 + 1$. Each block is a matrix as in (4.4), multiplied by $1/L$. As in Claim 4.2 we see that the smallest nonzero eigenvalue of this Hamiltonian is $c/LT_2^2 \geq c/T^2$ for some constant c . Now we can apply Lemma 3.1. This time, we apply it inside \mathcal{S}_{prop1} with

$$H_1 = (H_{out} + J_{in}H_{in})|_{\mathcal{S}_{prop1}}, \quad H_2 = J_2H'.$$

Eigenvectors of H_2 orthogonal to \mathcal{S}_{prop} have eigenvalue at least $J = J_2c/T^2$. As before, we can choose $J_2 = \text{poly}(N)$ such that $\lambda(H_1 + H_2)$ is lower bounded by $\lambda(H_1|_{\mathcal{S}_{prop}}) - \frac{1}{8}$. Hence, in the remainder we consider

$$H_{out}|_{\mathcal{S}_{prop}} + J_{in}H_{in}|_{\mathcal{S}_{prop}}.$$

4. *Restriction to \mathcal{S}_{in} .* The rest of the proof proceeds in the same way as in section 4. Indeed, the subspace \mathcal{S}_{prop} is isomorphic to the one in section 4 and both $H_{out}|_{\mathcal{S}_{prop}}$ and $H_{in}|_{\mathcal{S}_{prop}}$ are the same Hamiltonians. So by another application of Lemma 3.1 we get that the lowest eigenvalue of $H_{out}|_{\mathcal{S}_{prop}} + J_{in}H_{in}|_{\mathcal{S}_{prop}}$ is lower bounded by $\lambda(H_{out}|_{\mathcal{S}_{in}}) - \frac{1}{8}$. As in section 4, we have that $\lambda(H_{out}|_{\mathcal{S}_{in}}) > 1 - \varepsilon$ if the circuit accepts with probability less than ε . Hence $\lambda(H)$, the lowest eigenvalue of the original Hamiltonian H , is larger than $1 - \varepsilon - \frac{4}{8} = \frac{1}{2} - \varepsilon$. \square

6. Perturbation theory proof. In this section we give an alternative proof of our main theorem. In section 6.1, we develop our perturbation theory technique. Since this technique might constitute a useful tool in other Hamiltonian constructions, we keep the presentation as general as possible. Then, in section 6.2, we present a specific application of our technique, the three-qubit gadget. Finally, in section 6.3, we use this gadget to complete the proof of the main theorem.

6.1. Perturbation theory. The goal in perturbation theory is to analyze the spectrum of the sum of two Hamiltonians $\tilde{H} = H + V$ in the case that V has a small norm compared to the spectral gap of H . One setting was described in the projection lemma. Specifically, assume H has a zero eigenvalue with the associated eigenspace \mathcal{S} , whereas all other eigenvalues are greater than $\Delta \gg \|V\|$. The projection lemma shows that in this case, the lowest eigenvalue of \tilde{H} is close to that of $V|_{\mathcal{S}}$. In this section we find a better approximation to $\text{Spec } \tilde{H}$ by considering certain correction terms that involve higher powers of V . It turns out that these higher order correction terms include interesting interactions, which will allow us to create an effective 3-local Hamiltonian from 2-local terms. We remark that the projection lemma (for the entire lower part of the spectrum) can be obtained by following the development done in this section up to the first order.

Before giving a more detailed description of the technique, we need to introduce a certain amount of notation. For two Hermitian operators H and V , let $\tilde{H} = H + V$. We refer to H as the *unperturbed Hamiltonian* and to V as the *perturbation Hamiltonian*. Let $\lambda_j, |\psi_j\rangle$ be the eigenvalues and eigenvectors of H , whereas the eigenvalues and eigenvectors of \tilde{H} are denoted by $\tilde{\lambda}_j, |\tilde{\psi}_j\rangle$. In case of multiplicities, some eigenvalues might appear more than once. We order the eigenvalues in a nondecreasing order

$$\lambda_1 \leq \lambda_2 \leq \dots \leq \lambda_{\dim \mathcal{H}}, \quad \tilde{\lambda}_1 \leq \tilde{\lambda}_2 \leq \dots \leq \tilde{\lambda}_{\dim \mathcal{H}}.$$

In general, everything related to the perturbed Hamiltonian is marked with a tilde.

An important component in our proof is the *resolvent* of \tilde{H} , defined as

$$(6.1) \quad \tilde{G}(z) = (zI - \tilde{H})^{-1} = \sum_j (z - \tilde{\lambda}_j)^{-1} |\tilde{\psi}_j\rangle \langle \tilde{\psi}_j|.$$

It is a meromorphic⁸ operator-valued function of the complex variable z with poles at $z = \tilde{\lambda}_j$. In fact, for our purposes, it is sufficient to consider real z .⁹ Its usefulness comes from the fact that poles can be preserved under projections (while eigenvalues are usually lost). Similarly, we define the resolvent of H as $G(z) = (zI - H)^{-1}$.¹⁰

Let $\lambda_* \in \mathbb{R}$ be some cutoff on the spectrum of H .

DEFINITION 6.1. Let $\mathcal{H} = \mathcal{L}_+ \oplus \mathcal{L}_-$, where \mathcal{L}_+ is the space spanned by eigenvectors of H with eigenvalues $\lambda \geq \lambda_*$ and \mathcal{L}_- is spanned by eigenvectors of H of eigenvalue $\lambda < \lambda_*$. Let Π_{\pm} be the corresponding projection onto \mathcal{L}_{\pm} . For an operator X on \mathcal{H} define the operator $X_{++} = X|_{\mathcal{L}_+} = \Pi_+ X \Pi_+$ on \mathcal{L}_+ and similarly $X_{--} = X|_{\mathcal{L}_-}$. We also define $X_{+-} = \Pi_+ X \Pi_-$ as an operator from \mathcal{L}_- to \mathcal{L}_+ , and similarly X_{-+} .

With these definitions, in a representation of $\mathcal{H} = \mathcal{L}_+ \oplus \mathcal{L}_-$ both H and G are block-diagonal and we will omit one index for their blocks, i.e., $H_+ \stackrel{\text{def}}{=} H_{++}$,

⁸A meromorphic function is analytic in all but a discrete subset of \mathbb{C} , and these singularities must be poles and not essential singularities.

⁹The resolvent is the main tool in abstract spectral theory [19]. In physics, it is known as the *Green's function*. Physicists actually use slightly different Green's functions that are suited for specific problems.

¹⁰We can express \tilde{G} in terms of G (where we omit the variable z): $\tilde{G} = (G^{-1} - V)^{-1} = G(I - VG)^{-1} = G + GVG + GVGVG + GVGVGVG + \dots$. This expansion of \tilde{G} in powers of V may be represented by Feynman diagrams [1].

$G_+ \stackrel{\text{def}}{=} G_{++}$ and so on. Note that $G_{\pm}^{-1} = zI_{\pm} - H_{\pm}$. To summarize, we have

$$\begin{aligned} \tilde{H} &= \begin{pmatrix} \tilde{H}_{++} & \tilde{H}_{+-} \\ \tilde{H}_{-+} & \tilde{H}_{--} \end{pmatrix}, \quad V = \begin{pmatrix} V_{++} & V_{+-} \\ V_{-+} & V_{--} \end{pmatrix}, \quad H = \begin{pmatrix} H_+ & 0 \\ 0 & H_- \end{pmatrix}, \\ \tilde{G} &= \begin{pmatrix} \tilde{G}_{++} & \tilde{G}_{+-} \\ \tilde{G}_{-+} & \tilde{G}_{--} \end{pmatrix}, \quad G = \begin{pmatrix} G_+ & 0 \\ 0 & G_- \end{pmatrix}. \end{aligned}$$

We similarly write $\mathcal{H} = \tilde{\mathcal{L}}_+ \oplus \tilde{\mathcal{L}}_-$ according to the spectrum of \tilde{H} and the cutoff λ_* . Finally, we define

$$\Sigma_-(z) = zI_- - \tilde{G}_{--}^{-1}(z).$$

This operator-valued function is called *self-energy*.¹¹

With these notations in place, we can now give an overview of what follows. Our goal is to approximate the spectrum of $\tilde{H}|_{\tilde{\mathcal{L}}_-}$. We will do this by showing that in some sense, the spectrum of $\Sigma_-(z)$ gives such an approximation. To see why this arises, notice that by definition of $\Sigma_-(z)$, we have $\tilde{G}_{--}(z) = (zI_- - \Sigma_-(z))^{-1}$. In some sense, this equation is the analogue of (6.1), where $\Sigma_-(z)$ plays the role of a Hamiltonian for the projected resolvent $\tilde{G}_{--}(z)$. However, $\Sigma_-(z)$ is in general z -dependent and not a fixed Hamiltonian. Nonetheless, for certain choices of H and V , $\Sigma_-(z)$ is nearly constant in a certain range of z so we can choose an *effective Hamiltonian* H_{eff} that approximates $\Sigma_-(z)$ in this range. Our main theorem relates the spectrum of H_{eff} to that of $\tilde{H}|_{\tilde{\mathcal{L}}_-}$.

THEOREM 6.2. *Assume H has a spectral gap Δ around the cutoff λ_* ; i.e., all its eigenvalues are in $(-\infty, \lambda_-] \cup [\lambda_+, +\infty)$, where $\lambda_+ = \lambda_* + \Delta/2$ and $\lambda_- = \lambda_* - \Delta/2$. Assume, moreover, that $\|V\| < \Delta/2$. Let $\varepsilon > 0$ be arbitrary. Assume there exists an operator H_{eff} such that $\text{Spec } H_{\text{eff}} \subseteq [c, d]$ for some $c < d < \lambda_* - \varepsilon$ and, moreover, the inequality*

$$\|\Sigma_-(z) - H_{\text{eff}}\| \leq \varepsilon$$

holds for all $z \in [c - \varepsilon, d + \varepsilon]$. Then each eigenvalue $\tilde{\lambda}_j$ of $\tilde{H}|_{\tilde{\mathcal{L}}_-}$ is ε -close to the j th eigenvalue of H_{eff} .

The usefulness of the theorem comes from the fact that $\Sigma_-(z)$ has a natural series expansion, which can be truncated to obtain H_{eff} . This series may give rise to interesting terms; for example, in our application, 2-local terms in H and V lead to 3-local terms in H_{eff} . To obtain this expansion, we start by expressing \tilde{G} in terms of G as

$$\tilde{G} = (G^{-1} - V)^{-1} = \begin{pmatrix} G_+^{-1} - V_{++} & -V_{+-} \\ -V_{-+} & G_-^{-1} - V_{--} \end{pmatrix}^{-1}.$$

Then, using the block matrix identity

$$\begin{pmatrix} A & B \\ C & D \end{pmatrix}^{-1} = \begin{pmatrix} (A - BD^{-1}C)^{-1} & -A^{-1}B(D - CA^{-1}B)^{-1} \\ -D^{-1}C(A - BD^{-1}C)^{-1} & (D - CA^{-1}B)^{-1} \end{pmatrix}$$

¹¹As we will see later, this definition includes an H_- term. This term is usually not considered part of self-energy, but we have included it for notational convenience.

we conclude that

$$\tilde{G}_{--} = \left(G_{-}^{-1} - V_{--} - V_{-+}(G_{+}^{-1} - V_{++})^{-1}V_{+-} \right)^{-1}.$$

Finally, we can represent $\Sigma_{-}(z)$ using the series expansion $(I-X)^{-1} = I+X+X^2+\dots$,

$$\begin{aligned} \Sigma_{-}(z) &= H_{-} + V_{--} + V_{-+}(G_{+}^{-1} - V_{++})^{-1}V_{+-} \\ (6.2) \quad &= H_{-} + V_{--} + V_{-+}G_{+}(I_{+} - V_{++}G_{+})^{-1}V_{+-} \\ &= H_{-} + V_{--} + V_{-+}G_{+}V_{+-} + V_{-+}G_{+}V_{++}G_{+}V_{+-} \\ &\quad + V_{-+}G_{+}V_{++}G_{+}V_{++}G_{+}V_{+-} + \dots \end{aligned}$$

Proof of Theorem 6.2. We start with an overview of the proof. We first notice that, by definition, the eigenvalues of $\tilde{H}|_{\tilde{\mathcal{L}}_{-}}$ appear as poles in \tilde{G} . In Lemma 6.4, we show that these poles also appear as poles of \tilde{G}_{--} . As mentioned before, this is the reason we work with resolvents. In Lemmas 6.5 and 6.6 we relate these poles to the eigenvalues of Σ_{-} by showing that z is a pole of \tilde{G}_{--} if and only if it is an eigenvalue of $\Sigma_{-}(z)$. In other words, these are values of z for which $\Sigma_{-}(z)$ has z as an eigenvalue. Finally, we complete the proof of the theorem by using the assumption that $\Sigma_{-}(z)$ is close to H_{eff} , and thus any eigenvalue of $\Sigma_{-}(z)$ must be close to an eigenvalue of H_{eff} . This situation is illustrated in Figure 6.1.

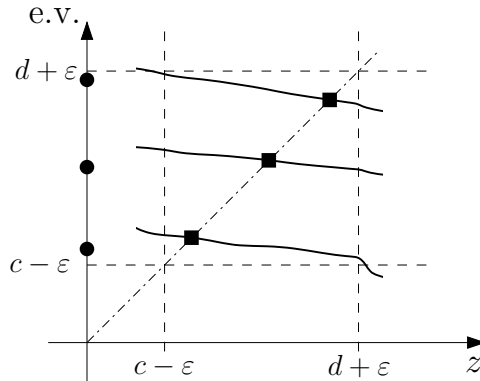


FIG. 6.1. The spectrum of $\Sigma_{-}(z)$ as a function of z is indicated with solid curves. The boxes correspond to the spectrum of $\tilde{H}|_{\tilde{\mathcal{L}}_{-}}$; they are those eigenvalues of $\Sigma_{-}(z)$ that lie on the dashed line $z = \text{e.v.}$ The dots indicate the spectrum of H_{eff} , which approximates the spectrum of $\tilde{H}|_{\tilde{\mathcal{L}}_{-}}$.

We start with a simple lemma that says that if two Hamiltonians H_1, H_2 are close, their spectra must also be close. It is a special case of Weyl’s inequalities (see, e.g., section III.2 in [6]).

LEMMA 6.3. *Let H_1, H_2 be two Hamiltonians with eigenvalues $\mu_1 \leq \mu_2 \leq \dots$ and $\sigma_1 \leq \sigma_2 \leq \dots$. Then, for all j , $|\mu_j - \sigma_j| \leq \|H_1 - H_2\|$.*

Proof. We will use a fact from the theory of Hermitian forms: If $X \leq Y$ (i.e., if $Y - X$ is positive semidefinite), then the operator Y has at least as many positive and nonnegative eigenvalues as X . Let $\varepsilon = \|H_1 - H_2\|$; then

$$(\mu_j - \varepsilon)I - H_2 \leq \mu_j I - H_1 \leq (\mu_j + \varepsilon)I - H_2.$$

The operator $\mu_j I - H_1$ has at most $j - 1$ positive and at least j nonnegative eigenvalues. Hence $(\mu_j - \varepsilon)I - H_2$ has at most $j - 1$ positive eigenvalues, and $(\mu_j + \varepsilon)I - H_2$ has at least j nonnegative eigenvalues. It follows that $\sigma_j \in [\mu_j - \varepsilon, \mu_j + \varepsilon]$. \square

The next lemma asserts that the poles of \tilde{G}_{--} in the range $(-\infty, \lambda_*)$ are in one-to-one correspondence with the eigenvalues of $\tilde{H}|_{\tilde{\mathcal{L}}_-}$. Hence we can recover the eigenvalues of $\tilde{H}|_{\tilde{\mathcal{L}}_-}$ from the poles of \tilde{G}_{--} .

LEMMA 6.4. *Let $\tilde{\lambda}$ be in $(-\infty, \lambda_*)$ and let $m \geq 0$ be its multiplicity as an eigenvalue of $\tilde{H}|_{\tilde{\mathcal{L}}_-}$. Then around $\tilde{\lambda}$, \tilde{G}_{--} is of the form $(z - \tilde{\lambda})^{-1}A + O(1)$, where A is a rank m operator.*

Proof. We first show that $\tilde{\mathcal{L}}_- \cap \mathcal{L}_+ = \{0\}$. Suppose the contrary; i.e., there is a nonzero vector $|\xi\rangle \in \tilde{\mathcal{L}}_- \cap \mathcal{L}_+$. Without loss of generality, $\langle \xi | \xi \rangle = 1$. Then we have $\langle \xi | (H + V) | \xi \rangle \leq \lambda_*$ (since $|\xi\rangle \in \tilde{\mathcal{L}}_-$) and $\langle \xi | H | \xi \rangle \geq \lambda_+$ (since $|\xi\rangle \in \mathcal{L}_+$). Hence $\langle \xi | V | \xi \rangle \leq \lambda_* - \lambda_+ = -\Delta/2$. But this is impossible because $\|V\| < \Delta/2$.

Now, since $\tilde{\mathcal{L}}_- \cap \mathcal{L}_+ = \{0\}$, we have that $\Pi_- |\xi\rangle \neq 0$ for all nonzero vectors $|\xi\rangle \in \tilde{\mathcal{L}}_-$. From (6.1) we obtain

$$\tilde{G}_{--} = \Pi_- \tilde{G} \Pi_- = \sum_j (z - \tilde{\lambda}_j)^{-1} \Pi_- |\tilde{\psi}_j\rangle \langle \tilde{\psi}_j | \Pi_-.$$

If the multiplicity of $\tilde{\lambda}$ is m , then the matrix $\sum |\tilde{\psi}_j\rangle \langle \tilde{\psi}_j |$ of the corresponding eigenvectors has rank m . This implies that the matrix $\sum \Pi_- |\tilde{\psi}_j\rangle \langle \tilde{\psi}_j | \Pi_-$ also has rank m . Indeed, if there is some linear combination of $\Pi_- |\tilde{\psi}_j\rangle$ that sums to zero, then taking the same linear combination of $|\tilde{\psi}_j\rangle$ must also sum to zero. \square

The next two lemmas relate the spectrum of $\tilde{H}|_{\tilde{\mathcal{L}}_-}$ to the operator $\Sigma_-(z)$.

LEMMA 6.5. *For any $z < \lambda_*$, the multiplicity of z as an eigenvalue of $\tilde{H}|_{\tilde{\mathcal{L}}_-}$ is equal to the multiplicity of z as an eigenvalue of $\Sigma_-(z)$.*

Proof. Fix some $z < \lambda_*$ and let m be its multiplicity as an eigenvalue of \tilde{H} (in particular, $m = 0$ if z is not an eigenvalue of \tilde{H}). In the neighborhood of z the function $\tilde{G}_{--}(w)$ has the form

$$\tilde{G}_{--}(w) = (w - z)^{-1}A + B + O(|w - z|),$$

where by Lemma 6.4, A is an operator of rank m . We now consider $\tilde{G}_{--}^{-1}(w)$. For any $w < \lambda_+ - \|V\|$ the norm of $G_+(w)$ is strictly less than $1/\|V\|$. Hence, by (6.2) we see that all the poles of $\Sigma_-(w)$ lie on the interval $[\lambda_+ - \|V\|, +\infty)$; in particular $\tilde{G}_{--}^{-1}(w) = wI_- - \Sigma_-(w)$ is analytic for $w \in (-\infty, \lambda_*)$. Hence we can write

$$\tilde{G}_{--}^{-1}(w) = wI_- - \Sigma_-(w) = C + D(w - z) + O(|w - z|^2).$$

We claim that the dimension of the null-space of C is exactly m . Notice that this implies that z is an m -fold eigenvalue of $\Sigma_-(z) = zI_- - C$. By multiplying the two equations above, we obtain

$$I_- = \tilde{G}_{--}^{-1}(w)\tilde{G}_{--}(w) = (w - z)^{-1}CA + (DA + CB) + O(|w - z|).$$

By equating coefficients, we obtain $CA = 0$ and $DA + CB = I_-$. On one hand, $CA = 0$ implies that the null-space of C has dimension at least m . On the other hand, the rank of DA is at most $\text{rank}(A) = m$. Since I_- has full rank, the dimension

of the null-space of CB must be at most m . This implies that the dimension of the null-space of C must also be at most m . \square

We observe that the function $\Sigma_-(z)$ is monotone decreasing in the operator sense (i.e., if $z_1 \leq z_2$, then $\Sigma_-(z_1) - \Sigma_-(z_2)$ is positive semidefinite):

$$\begin{aligned} \frac{d\Sigma_-(z)}{dz} &= \frac{d}{dz} \left(H_- + V_{--} + V_{-+}(zI_+ - H_+ - V_{++})^{-1}V_{+-} \right) \\ &= -V_{-+}(zI_+ - H_+ - V_{++})^{-2}V_{+-} \leq 0. \end{aligned}$$

LEMMA 6.6. *Let $\tilde{\lambda}_j$ be the j th eigenvalue of $\tilde{H}|_{\tilde{\mathcal{L}}_-}$. Then it is also the j th eigenvalue of $\Sigma_-(\tilde{\lambda}_j)$.*

Proof. For any $z \in \mathbb{R}$, let $f_1(z)$ (resp., $f_2(z)$) be the number of eigenvalues not greater than z of $\tilde{H}|_{\tilde{\mathcal{L}}_-}$ (resp., $\Sigma_-(z)$). When $z \rightarrow -\infty$, $f_1(z)$ is clearly 0. By the monotonicity of Σ_- we see that $f_2(z)$ is also 0. Using Lemma 6.5 we see that as z increases, both numbers increase together by the same amount m whenever z hits an eigenvalue of $\tilde{H}|_{\tilde{\mathcal{L}}_-}$ of multiplicity m (here we used again the monotonicity of Σ_-). Hence, for all z , $f_1(z) = f_2(z)$ and the lemma is proven. \square

We can now complete the proof of the theorem. By Lemma 6.3 and our assumption on H_{eff} , we have that for any $z \in [c - \varepsilon, d + \varepsilon]$, $\text{Spec } \Sigma_-(z)$ is contained in $[c - \varepsilon, d + \varepsilon]$. From this and the monotonicity of Σ_- , we obtain that there is no $z \in (d + \varepsilon, \lambda_*]$ that is an eigenvalue of $\Sigma_-(z)$. Similarly, there is no $z < c - \varepsilon$ that is an eigenvalue of $\Sigma_-(z)$. Hence, using Lemma 6.5 we see that $\text{Spec } \tilde{H}|_{\tilde{\mathcal{L}}_-}$ is contained in $[c - \varepsilon, d + \varepsilon]$. Now let $\tilde{\lambda}_j \in [c - \varepsilon, d + \varepsilon]$ be the j th eigenvalue of $\tilde{H}|_{\tilde{\mathcal{L}}_-}$. By Lemma 6.6 it is also the j th eigenvalue of $\Sigma_-(\tilde{\lambda}_j)$. By Lemma 6.3 it is ε -close to the j th eigenvalue of H_{eff} . \square

6.2. The three-qubit gadget. In this section we demonstrate how Theorem 6.2 can be used to transform a 3-local Hamiltonian into a 2-local one. The complete reduction will be shown in the next section. From now we try to keep the discussion more specialized to our QMA problem rather than presenting it in full generality as was done in section 6.1.

Let Y be some arbitrary 2-local Hamiltonian acting on a space \mathcal{M} of N qubits. Also, let B_1, B_2, B_3 be positive semidefinite Hamiltonians each acting on a different qubit (so they commute). We think of these four operators as having constant norm. Assume we have the 3-local Hamiltonian

$$(6.3) \quad Y - 6B_1B_2B_3.$$

The factor 6 is added for convenience. Recall that in the LOCAL HAMILTONIAN problem we are interested in the lowest eigenvalue of a Hamiltonian. Hence, our goal is to find a 2-local Hamiltonian whose lowest eigenvalue is very close to the lowest eigenvalue of (6.3).

We start by adding three qubits to our system. For $j = 1, 2, 3$, we denote the Pauli operators acting on the j th qubit by σ_j^α . Let $\delta > 0$ be a sufficiently small constant. Our 2-local Hamiltonian is $\tilde{H} = H + V$, where

$$\begin{aligned} H &= -\frac{\delta^{-3}}{4} I \otimes (\sigma_1^z \sigma_2^z + \sigma_1^z \sigma_3^z + \sigma_2^z \sigma_3^z - 3I), \\ V &= X \otimes I - \delta^{-2} (B_1 \otimes \sigma_1^x + B_2 \otimes \sigma_2^x + B_3 \otimes \sigma_3^x), \\ X &= Y + \delta^{-1} (B_1^2 + B_2^2 + B_3^2). \end{aligned}$$

The unperturbed Hamiltonian H has eigenvalues 0 and $\Delta \stackrel{def}{=} \delta^{-3}$. Associated with the zero eigenvalue is the subspace

$$\mathcal{L}_- = \mathcal{M} \otimes \mathcal{C}, \quad \text{where } \mathcal{C} = (|000\rangle, |111\rangle).$$

In the orthogonal subspace \mathcal{C}^\perp we have the states $|001\rangle, |010\rangle$, etc. We may think of the subspace \mathcal{C} as an effective qubit (as opposed to the three physical qubits); the corresponding Pauli operators are denoted by $\sigma_{\text{eff}}^\alpha$.

To obtain H_{eff} , we now compute the self-energy $\Sigma_-(z)$ using the power expansion in (6.2) up to the third order. There is no zeroth order term, i.e., $H_- = 0$. For the remaining terms, notice that $G_+ = (z - \Delta)^{-1} I_{\mathcal{C}_+}$. Hence, we can write $\Sigma_-(z)$ as

$$V_{--} + (z - \Delta)^{-1} V_{-+} V_{+-} + (z - \Delta)^{-2} V_{-+} V_{++} V_{+-} + (z - \Delta)^{-3} V_{-+} V_{++} V_{+-} + \dots$$

The first term is $V_{--} = X \otimes I_{\mathcal{C}}$ because a σ^x term takes any state in \mathcal{C} to \mathcal{C}^\perp . The expressions in the following terms are of the form

$$\begin{aligned} V_{-+} &= -\delta^{-2} (B_1 \otimes |000\rangle \langle 100| + B_2 \otimes |000\rangle \langle 010| + B_3 \otimes |000\rangle \langle 001| + \\ &\quad B_1 \otimes |111\rangle \langle 011| + B_2 \otimes |111\rangle \langle 101| + B_3 \otimes |111\rangle \langle 110|), \\ V_{++} &= X \otimes I_{\mathcal{C}^\perp} - \delta^{-2} (B_1 \otimes (|001\rangle \langle 101| + |010\rangle \langle 110| + |101\rangle \langle 001| + |110\rangle \langle 010|) + \\ &\quad B_2 \otimes (\dots) + B_3 \otimes (\dots)), \end{aligned}$$

where the dots denote similar terms for B_2 and B_3 . Now, in the second term of $\Sigma_-(z)$, V_{+-} flips one of the physical qubits, and V_{-+} must return it to its original state in order to return to the space \mathcal{C} . Hence we have $V_{-+} V_{+-} = \delta^{-4} (B_1^2 + B_2^2 + B_3^2) \otimes I_{\mathcal{C}}$. The third term is slightly more involved. Here we have two possible processes. In the first process, V_{+-} flips a qubit, V_{++} acts with $X \otimes I_{\mathcal{C}^\perp}$, and finally V_{-+} flips the qubit back. In the second process, V_{+-} , V_{++} , and V_{-+} flip all three qubits in succession. Thus,

$$\begin{aligned} (6.4) \quad \Sigma_-(z) &= X \otimes I_{\mathcal{C}} + (z - \Delta)^{-1} \delta^{-4} (B_1^2 + B_2^2 + B_3^2) \otimes I_{\mathcal{C}} \\ &\quad + (z - \Delta)^{-2} \delta^{-4} (B_1 X B_1 + B_2 X B_2 + B_3 X B_3) \otimes I_{\mathcal{C}} \\ &\quad - (z - \Delta)^{-2} \delta^{-6} \\ &\quad \quad (B_3 B_2 B_1 + B_2 B_3 B_1 + B_3 B_1 B_2 + B_1 B_3 B_2 + B_2 B_1 B_3 + B_1 B_2 B_3) \otimes \sigma_{\text{eff}}^x \\ &\quad + O(\|V\|^4 (z - \Delta)^{-3}). \end{aligned}$$

We now focus on the range $z = O(1) \ll \Delta$. In this range we have

$$(z - \Delta)^{-1} = -\frac{1}{\Delta} \left(1 - \frac{z}{\Delta}\right)^{-1} = -\frac{1}{\Delta} + O\left(\frac{z}{\Delta^2}\right) = -\delta^3 + O(\delta^6).$$

Simplifying, we obtain

$$\Sigma_-(z) = \underbrace{Y \otimes I_{\mathcal{C}} - 6B_1 B_2 B_3}_{H_{\text{eff}}} \otimes \sigma_{\text{eff}}^x + O(\delta).$$

Notice that $\|H_{\text{eff}}\| = O(1)$ and hence we obtain that for all z in, say, $[-2\|H_{\text{eff}}\|, 2\|H_{\text{eff}}\|]$ we have

$$\|\Sigma_-(z) - H_{\text{eff}}\| = O(\delta).$$

We may now apply Theorem 6.2 with $c = -\|H_{\text{eff}}\|$, $d = \|H_{\text{eff}}\|$, and $\lambda_* = \Delta/2$ to obtain the following result: Each eigenvalue $\tilde{\lambda}_j$ from the lower part of $\text{Spec } \tilde{H}$ is $O(\delta)$ -close to the j -th eigenvalue of H_{eff} . In fact, for our purposes, it is enough that the lowest eigenvalue of \tilde{H} is $O(\delta)$ -close to the lowest eigenvalue of H_{eff} . It remains to notice that the spectrum of H_{eff} consists of two parts that correspond to the effective spin states $|+\rangle = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$ and $|-\rangle = \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)$. Since $B_1 B_2 B_3$ is positive semidefinite, the smallest eigenvalue is associated with $|+\rangle$. Hence, the lowest eigenvalue of \tilde{H} is equal to the lowest eigenvalue of (6.3), as required.

6.3. Reduction from 3-Local Hamiltonian to 2-Local Hamiltonian.

In this section we reduce the 3-LOCAL HAMILTONIAN problem to the 2-LOCAL HAMILTONIAN problem. By the QMA-completeness of the 3-LOCAL HAMILTONIAN problem [11], this establishes Theorem 1.1.

THEOREM 6.7. *There is a polynomial time reduction from the 3-LOCAL HAMILTONIAN problem to the 2-LOCAL HAMILTONIAN problem.*

Proof. Recall that in the 3-LOCAL HAMILTONIAN problem (see Definition 2.3) we are given two constants a and b and a local Hamiltonian $H^{(3)} = \sum_j H_j$ such that each H_j is a three-qubit term whose norm is at most $\text{poly}(n)$. Our goal in this proof is to transform $H^{(3)}$ into a 2-local Hamiltonian $H^{(2)}$ whose lowest eigenvalue is close to that of $H^{(3)}$. We do this in two steps. The first is a somewhat technical step where we bring $H^{(3)}$ into a convenient form. In the second step, we replace each 3-local term with 2-local terms by using the gadget construction of the previous section. Before we continue with the proof, let us mention that it is crucial that we apply the gadget construction to all 3-local terms *simultaneously*. If instead we tried to apply the gadget construction sequentially, we would end up with an exponential blowup in the norms (since each application of the three-qubit gadget increases the norm by a multiplicative factor).

LEMMA 6.8. *The 3-local Hamiltonian $H^{(3)}$ can be represented as*

$$H^{(3)} = c_r \left(Y - 6 \sum_{m=1}^M B_{m1} B_{m2} B_{m3} \right),$$

where Y is a 2-local Hamiltonian with $\|Y\| = O(1/n^6)$, $M = O(n^3)$, each B_{mi} is a one-qubit term of norm $O(1/n^3)$ that satisfies $B_{mi} \geq \frac{1}{n^3} I$, and c_r is a rescaling factor satisfying $1 \leq c_r \leq \text{poly}(n)$.¹²

Proof. First, we can assume without loss of generality that each H_j acts on a different triple of qubits, and hence there are at most n^3 such terms. Recall that any three-qubit Hermitian operator can be written as a linear combination with real coefficients of the basis elements $\sigma^\alpha \otimes \sigma^\beta \otimes \sigma^\gamma$, where each of $\sigma^\alpha, \sigma^\beta, \sigma^\gamma$ ranges over the four possible Pauli matrices $\{I, \sigma^x, \sigma^y, \sigma^z\}$. Hence, for $M = O(n^3)$, we can write

$$H^{(3)} = c_r \left(-6 \sum_{m=1}^M c_m \cdot \sigma^{m,\alpha} \otimes \sigma^{m,\beta} \otimes \sigma^{m,\gamma} \right),$$

where each $\sigma^{m,\alpha}$ is a Pauli matrix acting on one of the qubits, and $c_r \leq \text{poly}(n)$ is chosen to be large enough so that $|c_m| \leq \frac{1}{n^9}$ for all $m = 1, \dots, M$.

¹²For the proof of Theorem 6.7 we need only the property $B_{mi} \geq 0$. The stronger property $B_{mi} \geq \frac{1}{n^3} I$ will be used in section 7.

We finish the proof by writing each $c_m \sigma^{m,\alpha} \otimes \sigma^{m,\beta} \otimes \sigma^{m,\gamma}$ as

$$\underbrace{\left(\frac{2}{n^3}I + n^6 c_m \sigma^{m,\alpha}\right)}_{B_{m1}} \otimes \underbrace{\left(\frac{2}{n^3}I + \frac{1}{n^3} \sigma^{m,\beta}\right)}_{B_{m2}} \otimes \underbrace{\left(\frac{2}{n^3}I + \frac{1}{n^3} \sigma^{m,\gamma}\right)}_{B_{m3}} + D_m,$$

where D_m is 2-local. Since $|c_m| \leq 1/n^9$ we have that $B_{mi} \geq \frac{1}{n^3}I$ and $\|D_m\| = O(1/n^9)$. \square

We now replace each term $-6B_{m1}B_{m2}B_{m3}$ by a three-qubit gadget. More specifically, let δ be a sufficiently small inverse polynomial in n to be chosen later. We consider the Hamiltonian $H^{(2)} = c_r \tilde{H}$, $\tilde{H} = H + V$, acting on a system of $n + 3M$ qubits, where

$$\begin{aligned} H &= -\frac{\delta^{-3}}{4} \sum_{m=1}^M I \otimes (\sigma_{m1}^z \sigma_{m2}^z + \sigma_{m1}^z \sigma_{m3}^z + \sigma_{m2}^z \sigma_{m3}^z - 3I), \\ V &= Y \otimes I + \delta^{-1} \sum_{m=1}^M (B_{m1}^2 + B_{m2}^2 + B_{m3}^2) \otimes I \\ (6.5) \quad &\quad - \delta^{-2} \sum_{m=1}^M (B_{m1} \otimes \sigma_{m1}^x + B_{m2} \otimes \sigma_{m2}^x + B_{m3} \otimes \sigma_{m3}^x). \end{aligned}$$

As before, let $\Delta = \delta^{-3}$ be the spectral gap of H . Notice that the spectrum of H includes not only 0 and Δ but also $2\Delta, 3\Delta, \dots, M\Delta$. Associated with the zero eigenvalue is the subspace spanned by all the zero-subspaces of the gadgets. Using $\|B_{mi}\| \leq O(1/n^3)$ and $M = O(n^3)$ we get $\|V\| = O(\delta^{-2}) < \Delta/2$.

The calculation of Σ_- is quite similar to the one-gadget case (cf. (6.4)). Each gadget contributes an independent term. Terms up to the third order can include only processes that involve one gadget. Indeed, in order to involve two gadgets, one has to flip a qubit from one gadget and from another gadget, and then flip both qubits back. Moreover, since only one gadget is involved, G_+ can be replaced by $(z - \Delta)^{-1} I_{\mathcal{L}_+}$ as before. From the fourth order onwards, processes start to include cross-terms between different gadgets. However, we claim that their contribution is only $O(\delta)$, as long as $|z| = O(1)$. Indeed, in this range, the eigenvalues of G_+ , which are $(z - \Delta)^{-1}, (z - 2\Delta)^{-1}, \dots$, are all at most $O(\delta^3)$ in absolute value, while the norm of each of the V terms is at most $O(\delta^{-2})$. To summarize, for $|z| = O(1)$,

$$(6.6) \quad \Sigma_-(z) = Y \otimes I_C - \underbrace{6 \sum_{m=1}^M B_{m1} B_{m2} B_{m3} \otimes (\sigma_m^x)_{\text{eff}}}_{H_{\text{eff}}} + O(\delta).$$

Since $\|H_{\text{eff}}\| \leq O(1)$, we can apply Theorem 6.2 with $c = -\|H_{\text{eff}}\|$, $d = \|H_{\text{eff}}\|$, and $\lambda_* = \Delta/2$. We obtain that the smallest eigenvalue of \tilde{H} is $O(\delta)$ -close to that of H_{eff} . The spectrum of H_{eff} consists of 2^M parts, corresponding to subspaces spanned by setting each effective spin state to either $|+\rangle$ or $|-\rangle$. Since $B_{m1}B_{m2}B_{m3} \geq 0$, the smallest eigenvalue of H_{eff} is achieved in the subspace where all effective spin states are in the $|+\rangle$ state. In this subspace, H_{eff} is identical to $H^{(3)}/c_r$. Hence, the smallest eigenvalue of $H^{(2)} = c_r \tilde{H}$ is $O(c_r \delta)$ -close to that of $H^{(3)}$. We complete the proof by choosing $\delta = c'/c_r$ for some small enough constant c' . \square

7. 2-local universal adiabatic computation. In this section we show that adiabatic computation with 2-local Hamiltonians is equivalent to “standard” quantum computation in the circuit model. In order to prove such an equivalence, one has to show that each model can simulate the other. One direction is already known: it is not too hard to show that any polynomial time adiabatic computation can be efficiently simulated by a quantum circuit [8]. Hence, it remains to show that adiabatic computation with 2-local Hamiltonians can efficiently simulate any quantum circuit. In [3] it is shown that adiabatic computation with 3-local Hamiltonians can efficiently simulate any quantum circuit. We obtain our result by combining their result with the techniques in our second proof.

Let us briefly mention the main ideas behind adiabatic computation. For more details see [3] and the references therein. In adiabatic computation, we consider a time-dependent Hamiltonian $H(s)$ for $s \in [0, 1]$ acting on a quantum system. We initialize the system in the groundstate of the initial Hamiltonian $H(0)$. This groundstate is required to be some simple quantum state that is easy to create. We then slowly modify the Hamiltonian from $s = 0$ to $s = 1$. We say that the adiabatic computation is *successful* if the final state of the system is close to the groundstate of $H(1)$. The adiabatic theorem (see, e.g., [18, 4]) says that if the Hamiltonian is modified slowly enough, the adiabatic computation is successful. In other words, it gives an upper bound on the running time of an adiabatic computation. For our purposes, it is enough to know that this bound is polynomial if for any $s \in [0, 1]$, the norm of $H(s)$, as well as those of its first and second derivatives, is bounded by a polynomial, and the spectral gap of $H(s)$ is larger than some inverse polynomial.

In [3] it is shown how to transform an arbitrary quantum circuit into an efficient 3-local adiabatic computation. To establish this, they define a 3-local time-dependent Hamiltonian $H^{(3)}(s)$ with the following properties. First, the Hamiltonian acts on a system of n qubits, where n is some constant times the number of gates in the circuit. Second, the groundstate of $H^{(3)}(0)$ is very easy to create (namely, it is the all zero state), and the groundstate of $H^{(3)}(1)$ is some state that encodes the result of the quantum circuit. Third, for all $s \in [0, 1]$, the spectral gap of $H^{(3)}(s)$ is bounded from below by an inverse polynomial in n , and the norm of $H^{(3)}(s)$, as well as those of its first and second derivatives, is bounded by some polynomial in n . Together with the adiabatic theorem, these properties imply that adiabatic computation according to $H^{(3)}(s)$ is efficient. Finally, let us mention that $H^{(3)}(s)$, as defined in [3], is linear in s ; that is, $H^{(3)}(s) = (1 - s)H^{(3)}(0) + sH^{(3)}(1)$. This property will be useful in our proof.

The following is the main theorem of this section.

THEOREM 7.1. *Any quantum computation can be efficiently simulated by an adiabatic computation with 2-local Hamiltonians.*

Proof. Given a quantum circuit, let $H^{(3)}(s)$ be the time-dependent Hamiltonian of [3] as described above. The idea of the proof is to apply the gadget construction of section 6.3 to $H^{(3)}(s)$ for any $s \in [0, 1]$, thereby creating a 2-local time-dependent Hamiltonian $H^{(2)}(s)$. Some care needs to be taken to ensure that the resulting time-dependent Hamiltonian is smooth enough as a function of s . We therefore describe how this is done in more detail.

We start by writing $H^{(3)}(s)$ in a form similar to that given by Lemma 6.8. Since $H^{(3)}(s)$ is linear in s , we can write

$$H^{(3)}(s) = c_r \left(-6 \sum_{m=1}^M c_m(s) \cdot \sigma^{m,\alpha} \otimes \sigma^{m,\beta} \otimes \sigma^{m,\gamma} \right),$$

where $M = O(n^3)$, each $c_m(s)$ is a linear function of s , and $c_r \leq \text{poly}(n)$ is chosen to be large enough so that $|c_m(s)| \leq \frac{1}{n^9}$ for all m and all $s \in [0, 1]$. Notice that c_r is a fixed scaling factor used for all $s \in [0, 1]$. Following the proof of Lemma 6.8, we write

$$H^{(3)}(s) = c_r \left(Y(s) - 6 \sum_{m=1}^M B_{m1}(s)B_{m2}B_{m3} \right),$$

where by our construction, $Y(s)$ and $B_{m1}(s)$ are linear in s , whereas B_{m2} and B_{m3} are independent of s . Finally, we define $H^{(2)}(s) = c_r \tilde{H}(s)$, where $\tilde{H}(s) = H + V(s)$ and the Hamiltonians H and $V(s)$ are defined as in (6.5). The parameter δ will be chosen later to be some small enough inverse polynomial in n .

In the rest of the proof, we show that adiabatic computation according to $H^{(2)}(s)$ can be used to simulate the given quantum circuit. We start by proving two lemmas that, together with the adiabatic theorem, imply that the running time of the adiabatic computation is polynomial in n .

LEMMA 7.2. *For any $s \in [0, 1]$, $\|H^{(2)}(s)\|$, $\|\frac{d}{ds}H^{(2)}(s)\|$, and $\|\frac{d^2}{ds^2}H^{(2)}(s)\|$ are upper bounded by a polynomial in n .*

Proof. Recall that $Y(s)$ and $B_{m1}(s)$ are linear in s . Together with the definition of $H^{(2)}$, this implies that $H^{(2)}(s)$ is a degree-two polynomial in s , i.e., we can write $H^{(2)}(s) = A + sB + s^2C$ for some Hermitian matrices A, B, C . It is not hard to see that the norm of each of these matrices is bounded by some polynomial in n . This implies that the norms of $H^{(2)}(s)$, its first derivative $B + 2sC$, and its second derivative $2C$ are bounded by some polynomial in n . \square

LEMMA 7.3. *For any $s \in [0, 1]$, the spectral gap of $H^{(2)}(s)$ is lower bounded by an inverse polynomial in n .*

Proof. As shown in section 6.3, the lower part of the spectrum of $H^{(2)}(s)$ is $O(c_r\delta)$ -close to the spectrum of $c_r H_{\text{eff}}(s)$. Hence, by choosing δ to be a small enough inverse polynomial in n , we see that it is enough to show that the spectral gap of $c_r H_{\text{eff}}(s)$ is at least some inverse polynomial in n .

The spectrum of $c_r H_{\text{eff}}(s)$ consists of 2^M parts, corresponding to all possible settings for the effective qubits. The part corresponding to the subspace in which all effective qubits are in the $|+\rangle$ state is identical to the spectrum of $H^{(3)}(s)$. Hence, we know that in this subspace the spectral gap is at least some inverse polynomial in n . We now claim that the lowest eigenvalue in all other $2^M - 1$ subspaces is greater than that in the all $|+\rangle$ subspace by at least some inverse polynomial in n . Indeed, the restriction of $c_r H_{\text{eff}}(s)$ to any such subspace is given by $H^{(3)}(s)$ plus a nonzero number of terms of the form $12c_r B_{m1}(s)B_{m2}B_{m3}$. The claim follows from the fact that $B_{m1}(s)B_{m2}B_{m3} \geq \frac{1}{n^9}I$. \square

To complete the proof, we need to argue about the groundstate of $H^{(2)}(0)$ and that of $H^{(2)}(1)$. To this end, we use the following lemma, which essentially says that if H_{eff} has a spectral gap, then Theorem 6.2 not only implies closeness in spectra but also in the groundstates.

LEMMA 7.4. *Assume that H, V, H_{eff} satisfy the conditions of Theorem 6.2 with some $\varepsilon > 0$. Let $\lambda_{\text{eff},i}$ denote the i th eigenvalue of H_{eff} and $|\tilde{v}\rangle$ (resp., $|v_{\text{eff}}\rangle$) denote the groundstate of \tilde{H} (resp., H_{eff}). Then, under the assumption $\lambda_{\text{eff},2} > \lambda_{\text{eff},1}$,*

$$|\langle \tilde{v} | v_{\text{eff}} \rangle| \geq 1 - \frac{2\|V\|^2}{(\lambda_+ - \lambda_{\text{eff},1} - \varepsilon)^2} - \frac{4\varepsilon}{\lambda_{\text{eff},2} - \lambda_{\text{eff},1}}.$$

Before we prove the lemma, let us complete the proof of the theorem. Recall that in our case $\varepsilon = O(\delta)$, $\|V\| = O(\delta^{-2})$, $\lambda_+ = \delta^{-3}$, $|\lambda_{\text{eff},1}| \leq O(1)$, and $\lambda_{\text{eff},2} - \lambda_{\text{eff},1} = 1/\text{poly}(n)$. Hence, the first error term in the above bound is $O(\delta^2)$ while the second is $O(\delta \cdot \text{poly}(n))$. Therefore, by choosing δ to be a small enough inverse polynomial in n , we can guarantee that the groundstate of $H^{(2)}(s)$ is close to the groundstate of $H_{\text{eff}}(s)$. In particular, the groundstate of $H^{(2)}(1)$, which is the output of the adiabatic computation, is close to the groundstate of $H_{\text{eff}}(1)$. The latter is $|v_1\rangle \otimes |+\rangle^{\otimes M}$, where $|v_1\rangle$ is the groundstate of $H^{(3)}(1)$. By simply tracing out the $3M$ gadget qubits, we can recover $|v_1\rangle$ from this groundstate and therefore obtain the output of the quantum circuit. Similarly, the groundstate of $H^{(2)}(0)$, which is the state to which the system should be initialized, is close to the groundstate of $H_{\text{eff}}(0)$. The latter is $|v_0\rangle \otimes |+\rangle^{\otimes M}$, where $|v_0\rangle$ is the groundstate of $H^{(3)}(0)$. We therefore initialize the system by setting the original n qubits to $|v_0\rangle$ and the M gadgets to the effective $|+\rangle$ state. This state is close to the groundstate of $H^{(2)}(0)$, and since the adiabatic computation is unitary, this approximation does not affect the output by much.

It remains to prove the lemma.

Proof of Lemma 7.4. Let $|\tilde{v}_-\rangle = \Pi_- |\tilde{v}\rangle / \|\Pi_- |\tilde{v}\rangle\|$ be the normalized projection of $|\tilde{v}\rangle$ on the space \mathcal{L}_- . We first show that $|\tilde{v}_-\rangle$ is close to $|\tilde{v}\rangle$. By Theorem 6.2, we know that $\tilde{\lambda}_1 \leq \lambda_{\text{eff},1} + \varepsilon$. Hence,

$$\|\Pi_+ \tilde{H} |\tilde{v}\rangle\| = \tilde{\lambda}_1 \|\Pi_+ |\tilde{v}\rangle\| \leq (\lambda_{\text{eff},1} + \varepsilon) \|\Pi_+ |\tilde{v}\rangle\|$$

and

$$\|\Pi_+ \tilde{H} |\tilde{v}\rangle\| = \|\Pi_+ H |\tilde{v}\rangle + \Pi_+ V |\tilde{v}\rangle\| \geq \|\Pi_+ H |\tilde{v}\rangle\| - \|V\| \geq \lambda_+ \|\Pi_+ |\tilde{v}\rangle\| - \|V\|.$$

By combining the two inequalities we obtain

$$\|\Pi_+ |\tilde{v}\rangle\| \leq \frac{\|V\|}{\lambda_+ - \lambda_{\text{eff},1} - \varepsilon},$$

from which we see that

$$\alpha \stackrel{\text{def}}{=} |\langle \tilde{v} | \tilde{v}_- \rangle| = \|\Pi_- |\tilde{v}\rangle\| \geq \|\Pi_- |\tilde{v}\rangle\|^2 \geq 1 - \frac{\|V\|^2}{(\lambda_+ - \lambda_{\text{eff},1} - \varepsilon)^2}.$$

Our next step is to show that $|\tilde{v}_-\rangle$ is close to $|v_{\text{eff}}\rangle$. For this we need to consider the proof of Theorem 6.2. We start by taking Lemma 6.4 with $\tilde{\lambda} = \tilde{\lambda}_1$. The lemma says that A is a matrix of rank 1. By looking at the proof, it is easy to see that A is in fact $\Pi_- |\tilde{v}\rangle \langle \tilde{v} | \Pi_-$. Next, Lemma 6.5 implies that $\tilde{\lambda}_1$ is an eigenvalue of multiplicity 1 of $\Sigma_-(\tilde{\lambda}_1)$. In fact, from the proof it follows that the corresponding eigenvector is exactly $\Pi_- |\tilde{v}\rangle$ (since the null space of C is equal to the span of A). By normalizing, this is exactly $|\tilde{v}_-\rangle$. But by our assumption, $\|\Sigma_-(z) - H_{\text{eff}}\| \leq \varepsilon$ for all $z \in [c - \varepsilon, d + \varepsilon]$ and in particular

$$\|\Sigma_-(\tilde{\lambda}_1) - H_{\text{eff}}\| \leq \varepsilon.$$

From this we obtain that

$$|\langle \tilde{v}_- | (\Sigma_-(\tilde{\lambda}_1) - H_{\text{eff}}) | \tilde{v}_- \rangle| \leq \varepsilon$$

and hence

$$\langle \tilde{v}_- | H_{\text{eff}} | \tilde{v}_- \rangle \leq \tilde{\lambda}_1 + \varepsilon \leq \lambda_{\text{eff},1} + 2\varepsilon,$$

where we again used that $\tilde{\lambda}_1 \leq \lambda_{\text{eff},1} + \varepsilon$. Since H_{eff} has a spectral gap, this indicates that $|\tilde{v}_-\rangle$ must be close to $|v_{\text{eff}}\rangle$. Indeed, let $\beta = |\langle \tilde{v}_- | v_{\text{eff}} \rangle|$. Then,

$$\langle \tilde{v}_- | H_{\text{eff}} | \tilde{v}_- \rangle \geq \beta^2 \lambda_{\text{eff},1} + (1 - \beta^2) \lambda_{\text{eff},2} = \lambda_{\text{eff},1} + (1 - \beta^2)(\lambda_{\text{eff},2} - \lambda_{\text{eff},1}).$$

By combining the two inequalities we obtain

$$1 - \beta^2 \leq \frac{2\varepsilon}{\lambda_{\text{eff},2} - \lambda_{\text{eff},1}}.$$

Summarizing,

$$\begin{aligned} |\langle \tilde{v} | v_{\text{eff}} \rangle| &= |\langle \tilde{v} | \tilde{v}_- \rangle \langle \tilde{v}_- | v_{\text{eff}} \rangle + \langle \tilde{v} | (I - |\tilde{v}_-\rangle \langle \tilde{v}_-|) | v_{\text{eff}} \rangle| \\ &\geq \alpha \cdot \beta - \sqrt{(1 - \alpha^2)(1 - \beta^2)} \geq \alpha \cdot \beta - \frac{1}{2}((1 - \alpha^2) + (1 - \beta^2)) \\ &\geq (1 - (1 - \alpha) - (1 - \beta)) - ((1 - \alpha) + (1 - \beta)) = 1 - 2(1 - \alpha) - 2(1 - \beta) \\ &\geq 1 - \frac{2\|V\|^2}{(\lambda_+ - \lambda_{\text{eff},1} - \varepsilon)^2} - \frac{4\varepsilon}{\lambda_{\text{eff},2} - \lambda_{\text{eff},1}}. \quad \square \end{aligned}$$

8. Conclusion. Some interesting open questions remain. First, perturbation theory has allowed us to perform the first reduction *inside* QMA. What other problems can be solved using this technique? Second, there exists an intriguing class between NP (in fact, MA) and QMA known as QCMA. It is the class of problems that can be verified by a quantum verifier with a *classical* proof. Can one show a separation between QCMA and QMA, or perhaps show that they are equal? Third, Kitaev's original 5-local proof has the following desirable property: For any YES instance produced by the reduction there exists a state such that each individual 5-local term is very close to its groundstate. Note that this is a stronger property than the one required in the LOCAL HAMILTONIAN problem. Using a slight modification of Kitaev's original construction, one can show a reduction to the 4-LOCAL HAMILTONIAN problem that has the same property. However, we do not know if this property can be achieved for the 3-local or the 2-local problem.

Acknowledgments. Discussions with Sergey Bravyi and Frank Verstraete are gratefully acknowledged.

REFERENCES

- [1] A. A. ABRIKOSOV, L. P. GORKOV, AND I. E. DZIALOSHINSKI, *Methods of Quantum Field Theory in Statistical Physics*, Dover, New York, 1975.
- [2] D. AHARONOV AND T. NAVEH, *Quantum NP—A survey*, 2002; available online from <http://www.arxiv.org/abs/quant-ph/0210077>.
- [3] D. AHARONOV, W. VAN DAM, J. KEMPE, Z. LANDAU, S. LLOYD, AND O. REGEV, *Adiabatic quantum computation is equivalent to standard quantum computation*, in Proceedings of the 45th Annual IEEE Symposium on Foundations of Computer Science, 2004, pp. 42–51.
- [4] A. AMBAINIS AND O. REGEV, *An elementary proof of the quantum adiabatic theorem*, 2004; available online from <http://www.arxiv.org/abs/quant-ph/0411152>.
- [5] D. BARENCO, C. H. BENNETT, R. CLEVE, D. P. DIVINCENZO, N. MARGOLUS, P. SHOR, T. SLEATOR, J. SMOLIN, AND H. WEINFURTER, *Elementary gates for quantum computation*, Phys. Rev. A, 52 (1995), pp. 3457–3467.
- [6] R. BHATIA, *Matrix Analysis*, Grad. Texts in Math. 169, Springer-Verlag, New York, 1997.
- [7] S. BRAVYI AND M. VYALYI, *Commutative version of the k-local Hamiltonian problem and non-triviality check for quantum codes*, Quantum Inf. Comput., 5 (2005), pp. 187–215.
- [8] E. FARHI, J. GOLDSTONE, S. GUTMANN, AND M. SIPSER, *Quantum computation by adiabatic evolution*, 2000; available online from <http://www.arxiv.org/abs/quant-ph/0001106>.

- [9] D. JANZING, P. WOCJAN, AND T. BETH, “*Non-identity-check*” is QMA-complete, *Internat. J. Quantum Inform.*, 3 (2005), pp. 463–473.
- [10] J. KEMPE, A. KITAEV, AND O. REGEV, *The complexity of the local Hamiltonian problem*, in Proceedings of the 24th Annual FSTTCS, Lecture Notes in Comput. Sci. 3328, Springer-Verlag, Berlin, 2004, pp. 372–383.
- [11] J. KEMPE AND O. REGEV, *3-local Hamiltonian is QMA-complete*, *Quantum Inf. Comput.*, 3 (2003), pp. 258–264.
- [12] A. Y. KITAEV, A. H. SHEN, AND M. N. VYALYI, *Classical and quantum computation*, Grad. Stud. Math. 47, AMS, Providence, RI, 2002.
- [13] E. KNILL, *Quantum randomness and nondeterminism*, 1996; available online from <http://www.arxiv.org/abs/quant-ph/9610012>.
- [14] C. MARRIOTT AND J. WATROUS, *Quantum Arthur–Merlin games*, in Proceedings of the 19th IEEE Annual Conference on Computational Complexity (CCC), 2004, pp. 344–354.
- [15] M. NIELSEN AND I. CHUANG, *Quantum Computation and Quantum Information*, Cambridge University Press, Cambridge, UK, 2000.
- [16] R. OLIVEIRA AND B. TERHAL, *The complexity of quantum spin systems on a two-dimensional square lattice*, 2005; available online from <http://www.arxiv.org/abs/quant-ph/0504050>.
- [17] C. PAPADIMITRIOU, *Computational Complexity*, Addison–Wesley, Reading, MA, 1994.
- [18] B. REICHARDT, *The quantum adiabatic optimization algorithm and local minima*, in Proceedings of the 36th Annual STOC, ACM, 2004, pp. 502–510.
- [19] W. RUDIN, *Functional Analysis*, 2nd ed., International Series in Pure and Applied Mathematics, McGraw–Hill Inc., New York, 1991.
- [20] J. WATROUS, *Succinct quantum proofs for properties of finite groups*, in Proceedings of the 41st Annual IEEE Symposium on Foundations of Computer Science, 2000, pp. 537–546.
- [21] P. WOCJAN AND T. BETH, *The 2-local Hamiltonian problem encompasses NP*, *Internat. J. Quantum Inform.*, 1 (2003), pp. 349–357.

ALGORITHMS FOR COMBINING ROOTED TRIPLETS INTO A GALLED PHYLOGENETIC NETWORK*

JESPER JANSSON[†], NGUYEN BAO NGUYEN[†], AND WING-KIN SUNG^{†‡}

Abstract. This paper considers the problem of determining whether a given set \mathcal{T} of rooted triplets can be merged without conflicts into a galled phylogenetic network and, if so, constructing such a network. When the input \mathcal{T} is dense, we solve the problem in $O(|\mathcal{T}|)$ time, which is optimal since the size of the input is $\Theta(|\mathcal{T}|)$. In comparison, the previously fastest algorithm for this problem runs in $O(|\mathcal{T}|^2)$ time. We also develop an optimal $O(|\mathcal{T}|)$ -time algorithm for enumerating all simple phylogenetic networks leaf-labeled by L that are consistent with \mathcal{T} , where L is the set of leaf labels in \mathcal{T} , which is used by our main algorithm. Next, we prove that the problem becomes NP-hard if extended to nondense inputs, even for the special case of simple phylogenetic networks. We also show that for every positive integer n , there exists some set \mathcal{T} of rooted triplets on n leaves such that any galled network can be consistent with at most $0.4883 \cdot |\mathcal{T}|$ of the rooted triplets in \mathcal{T} . On the other hand, we provide a polynomial-time approximation algorithm that always outputs a galled network consistent with at least a factor of $\frac{5}{12}$ (> 0.4166) of the rooted triplets in \mathcal{T} .

Key words. phylogenetic network, galled network, rooted triplet, *SN*-tree, NP-hardness, algorithm

AMS subject classifications. 68Q25, 68R10, 05C85, 92B99

DOI. 10.1137/S0097539704446529

1. Introduction. A *rooted triplet* is a binary, rooted, unordered tree with three distinctly labeled leaves. Aho et al. [1] introduced the problem of determining whether a given set of rooted triplets can be combined without conflicts into a distinctly leaf-labeled tree which contains each of the given rooted triplets as an embedded subtree, and, if so, returning one. The original motivation for this problem came from an application in the theory of relational databases (see [1] for details), but it has since been studied further and generalized because of its applications to phylogenetic tree construction [2, 6, 7, 10, 13, 14, 17, 20, 21, 23, 25]. Here, we study an extension of the problem in which the objective is to determine if a given set \mathcal{T} of rooted triplets can be merged into a more complex structure known as a *galled phylogenetic network*.

A *phylogenetic network* is a type of distinctly leaf-labeled, directed acyclic graph that can be used to model nontreelike evolution. A number of methods for inferring phylogenetic networks under various assumptions and using different kinds of data have been proposed recently [8, 12, 15, 19, 22, 24]. A *galled phylogenetic network*, or *galled network* for short, is an important, biologically motivated structural restriction of a phylogenetic network (see section 1.2) in which all cycles in the underlying undirected graph are node-disjoint.¹

*Received by the editors November 12, 2004; accepted for publication (in revised form) October 2, 2005; published electronically March 3, 2006. An extended abstract of this article has appeared in *Proceedings of the 16th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, 2005, pp. 349–358.

<http://www.siam.org/journals/sicomp/35-5/44652.html>

[†]School of Computing, National University of Singapore, 3 Science Drive 2, 117543 Singapore (jansson@comp.nus.edu.sg, nguyeba@comp.nus.edu.sg, ksung@comp.nus.edu.sg). The first author was supported in part by Kyushu University and JSPS (Japan Society for the Promotion of Science).

[‡]Genome Institute of Singapore, 60 Biopolis Street, Genome, 138672 Singapore.

¹Without the node-disjoint constraint, our problem becomes trivial to solve since then a solution always exists and, furthermore, can be obtained in polynomial time using a simple sorting network-based construction [15].

We present several new results for the problem of inferring a galled network consistent with a given set \mathcal{T} of rooted triplets. Denote the set of leaf labels in \mathcal{T} by L . If \mathcal{T} contains at least one rooted triplet for each cardinality-three subset of L , then \mathcal{T} is called *dense*. We first give an exact algorithm named *FastGalledNetwork* for dense inputs whose running time is $O(|\mathcal{T}|)$. In comparison, the previously fastest known algorithm for this case runs in $O(|\mathcal{T}|^2)$ time [15]. Since the size of the input is $\Theta(|\mathcal{T}|)$ when \mathcal{T} is dense and any algorithm that solves the problem must look at the entire input, the asymptotic running time of our new algorithm is optimal. The improvement in running time is due to two observations: first, that the so-called *SN*-sets employed in [15] do not have to be explicitly computed but can be represented using a tree (the *SN*-tree) which we can construct in $O(|\mathcal{T}|)$ time, and second, that the *SN*-tree can be expanded into a galled network consistent with \mathcal{T} (if one exists) in $O(|\mathcal{T}|)$ time by replacing each internal node of degree 3 or higher with a special kind of network found by applying an algorithm called *SimpleNetworks*.

Next, we show that the problem becomes NP-hard when \mathcal{T} is not required to be dense by giving a polynomial-time reduction from Set Splitting. Finally, we consider approximation algorithms. We present an $O(|L| \cdot |\mathcal{T}|^3)$ -time algorithm that always outputs a galled network consistent with at least a factor of $\frac{5}{12}$ (> 0.4166) of the rooted triplets in \mathcal{T} for any \mathcal{T} . (Our approximation algorithm can also be applied in the dense case when the input cannot be combined into a galled network without conflicts.) On the negative side, we show that there exist inputs for which any galled network can be consistent with at most a factor of 0.4883 of the rooted triplets. It is interesting to note that for *trees*, the corresponding bounds are known to be tight [7]; that is, there is a polynomial-time approximation algorithm which always constructs a tree consistent with at least $\frac{1}{3} \cdot |\mathcal{T}|$ of the rooted triplets in \mathcal{T} , and there exist some inputs for which no tree can achieve a factor higher than $\frac{1}{3} \cdot |\mathcal{T}|$.

1.1. Definitions and notation. A *phylogenetic tree* is a binary, rooted, unordered tree whose leaves are distinctly labeled. A *phylogenetic network* is a generalization of a phylogenetic tree formally defined as a rooted, connected, directed acyclic graph in which (1) exactly one node has indegree 0 (the *root*), and all other nodes have indegree 1 or 2; (2) all nodes with indegree 2 (referred to as *hybrid nodes*) have outdegree 1, and all other nodes have outdegree 0 or 2; and (3) all nodes with outdegree 0 (the *leaves*) are distinctly labeled. For any phylogenetic network N , let $\mathcal{U}(N)$ be the undirected graph obtained from N by replacing each directed edge by an undirected edge. N is said to be a *galled phylogenetic network* (*galled network*, for short) if all cycles in $\mathcal{U}(N)$ are node-disjoint. Galled networks are also known in the literature as *topologies with independent recombination events* [24], *galled-trees* [8], *gt-networks* [19], and *level-1 phylogenetic networks* [4, 15].

A phylogenetic tree with exactly three leaves is called a *rooted triplet*. The unique rooted triplet on a leaf set $\{x, y, z\}$ in which the lowest common ancestor of x and y is a proper descendant of the lowest common ancestor of x and z (or, equivalently, where the lowest common ancestor of x and y is a proper descendant of the lowest common ancestor of y and z) is denoted by $(\{x, y\}, z)$. For any phylogenetic network N , a rooted triplet $t = (\{x, y\}, z)$ is said to be *consistent with N* if t is an embedded subtree of N (i.e., if a lowest common ancestor of x and y in N is a proper descendant of a lowest common ancestor of x and z in N), and a set \mathcal{T} of rooted triplets is said to be *consistent with N* if every rooted triplet in \mathcal{T} is consistent with N .

Denote the set of leaves in any phylogenetic network N by $\Lambda(N)$ and for any set \mathcal{T} of rooted triplets, define $\Lambda(\mathcal{T}) = \bigcup_{t_i \in \mathcal{T}} \Lambda(t_i)$. A set \mathcal{T} of rooted triplets is *dense* if

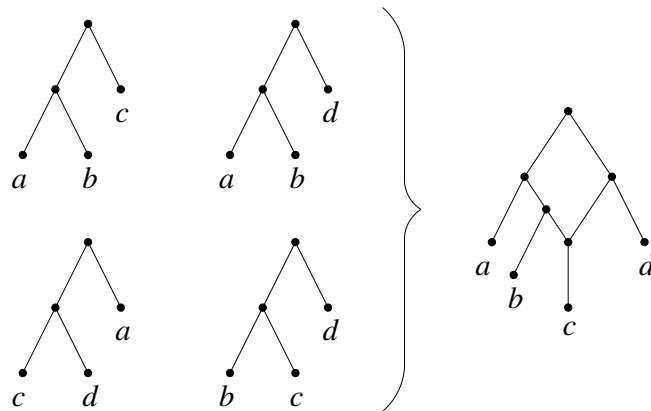


FIG. 1. A dense set \mathcal{T} of rooted triplets with leaf set $\{a, b, c, d\}$ and a galled phylogenetic network which is consistent with \mathcal{T} . Note that this solution is not unique.

for each $\{x, y, z\} \subseteq \Lambda(\mathcal{T})$, at least one of $(\{x, y\}, z)$, $(\{x, z\}, y)$, and $(\{y, z\}, x)$ belongs to \mathcal{T} . If \mathcal{T} is dense, then $|\mathcal{T}| = \Theta(|\Lambda(\mathcal{T})|^3)$. Furthermore, for any set \mathcal{T} of rooted triplets and $L' \subseteq \Lambda(\mathcal{T})$, define $\mathcal{T}|L'$ as the subset of \mathcal{T} consisting of all rooted triplets t with $\Lambda(t) \subseteq L'$. The problem we consider here is the following: Given a set \mathcal{T} of rooted triplets, output a galled network N with $\Lambda(N) = \Lambda(\mathcal{T})$ such that N and \mathcal{T} are consistent if such a network exists; otherwise, output *null*. See Figure 1 for an example. Throughout this paper, we write $L = \Lambda(\mathcal{T})$ and $n = |L|$.

To describe our algorithms, we need the following additional terminology. Let N be a phylogenetic network. We call nodes with indegree 2 *hybrid nodes* and their parent edges *hybrid edges*. Let h be a hybrid node in N . Every ancestor s of h such that h can be reached using two disjoint directed paths starting at the children of s is called a *split node of h* . If s is a split node of h , then any path starting at s and ending at h is called a *merge path of h* , and any path starting at a child of s and ending at a parent of h is called a *clipped merge path of h* . From the above, it follows that in a galled network, each split node is a split node of exactly one hybrid node, and each hybrid node has exactly one split node.

Let N be a galled network. For any node u in N , $N[u]$ denotes the subnetwork of N rooted at u , i.e., the minimal subgraph of N which includes all nodes and directed edges of N reachable from u . $N[u]$ is called a *side network* of N if there exists a merge path P in N such that u does not belong to P but u is a child of a node belonging to P . In this case, $N[u]$ is also said to be *attached to P* . N is called a *simple phylogenetic network* (or *simple network*) if N has exactly one hybrid node h , the root node of N is the split node of h , and every side network of N is a leaf. For example, the galled network on the right in Figure 1 is a simple network. For any simple network N , denote the leaf attached to the hybrid node in N by $hl(N)$.

1.2. Motivation. Phylogenetic networks are used by scientists to describe evolutionary relationships that do not fit the traditional models in which evolution is assumed to be treelike. Evolutionary events such as horizontal gene transfer or hybrid speciation (often referred to as *recombination events*) cannot be adequately represented in a single tree [8, 9, 19, 22, 24] but can be modeled in a phylogenetic network as internal nodes having more than one parent. Galled networks are an important type of phylogenetic network and have attracted special attention in the

literature [4, 8, 19, 24] due to their biological significance (see [8] for a discussion) and their simple, almost treelike, structure. When the number of recombination events is limited and most of them have occurred recently, a galled network may suffice to accurately describe the evolutionary process under study [8].

A challenge in the field of phylogenetics is to develop efficient and reliable methods for constructing and comparing phylogenetic networks. For example, to construct a meaningful phylogenetic network for a large subset of the human population (which may subsequently be used to help locate regions in the genome associated with some observable trait indicating a particular disease) in the future, efficient algorithms are crucial because the input can be expected to be very large. The motivation behind the rooted triplet approach taken in this paper is that a highly accurate tree for each cardinality-three subset of the leaf set can be obtained through maximum likelihood-based methods such as [3] or Sibley–Ahlquist-style DNA–DNA hybridization experiments (see [17]). Hence, the algorithms presented in [15] and here can be used as the merging step in a divide-and-conquer approach for constructing phylogenetic networks analogous to the quartet method paradigm for inferring unrooted phylogenetic trees [16, 18] and other supertree methods (see [10, 21] and the references therein). We consider dense input sets in particular since this case can be solved in polynomial time.

1.3. Related work. Aho et al. [1] gave an $O(|\mathcal{T}| \cdot n)$ -time algorithm for determining whether a given set \mathcal{T} of rooted triplets on n leaves is consistent with some rooted, distinctly leaf-labeled tree, and, if so, returning such a tree. Henzinger, King, and Warnow [10] improved its running time to $\min\{O(|\mathcal{T}| \cdot n^{0.5}), O(|\mathcal{T}| + n^2 \log n)\}$; in fact, replacing the deterministic algorithm for dynamic graph connectivity employed by Henzinger, King, and Warnow, with a more recent one due to Holm, de Lichtenberg, and Thorup [11] yields a running time of $\min\{O(|\mathcal{T}| \cdot \log^2 n), O(|\mathcal{T}| + n^2 \log n)\}$ [14]. Gąsieniec et al. [6] studied a variant of the problem for *ordered* trees. Ng and Wormald [21] considered the problem of constructing *all* rooted, unordered trees distinctly leaf-labeled by $\Lambda(\mathcal{T})$ that are consistent with \mathcal{T} .

If two or more of the rooted triplets are in conflict, i.e., contain contradicting branching information, the algorithm of Aho et al. returns a null tree. However, this is not very practical in certain applications. For example, in the context of constructing a phylogenetic tree from a set of rooted triplets, some errors may occur in the input when the rooted triplets are based on data obtained experimentally, yet a nonnull tree is still required. In this case, one can try to construct a tree consistent with the maximum number of rooted triplets in the input [2, 6, 7, 13, 25], or a tree with as many leaves from $\Lambda(\mathcal{T})$ as possible which is consistent with all input rooted triplets involving these leaves only [14]. Although the former problem is NP-hard [2, 13, 25], Gąsieniec et al. [7] showed that it has a polynomial-time approximation algorithm that outputs a distinctly leaf-labeled tree consistent with at least $\frac{1}{3}$ of the given rooted triplets, which is a tight bound in the sense that there exist inputs \mathcal{T} such that any distinctly leaf-labeled tree can be consistent with at most $\frac{1}{3}$ of the rooted triplets in \mathcal{T} .²

The problem studied in this paper was introduced in [15]. The main result of [15] is an exact $O(|\mathcal{T}|^2)$ -time algorithm for the dense case. Reference [15] also showed that if no restrictions are placed on the structure of the output phylogenetic network (i.e., if nongalled networks are allowed), then the problem always has a solution which

²On the other hand, if the optimal solution contains a large fraction of the input rooted triplets, another approximation presented in [7] (based on minimum cuts in the auxiliary graph introduced by Aho et al. [1]) gives a better approximation factor.

can be easily obtained from any given sorting network for n elements. Nakhleh et al. [19] gave an $O(n^2)$ -time algorithm for the related problem of determining if two given phylogenetic trees T_1 and T_2 with identical leaf sets can be combined into a galled network containing both T_1 and T_2 as embedded subtrees and, if so, constructing one with the smallest possible number of hybrid nodes, where n is the number of leaves (in fact, this is equivalent to inferring a galled network consistent with the set of all rooted triplets which are embedded subtrees of T_1 or T_2). They also studied the case where T_1 and T_2 may contain errors but only one hybrid node is allowed. Huson et al. [12] considered a similar problem for constructing an *unrooted* phylogenetic network from a set of unrooted, distinctly leaf-labeled trees.

1.4. Organization of the paper. In section 2, we present a new algorithm called *SimpleNetworks* for computing all simple phylogenetic networks consistent with a given dense set \mathcal{T} of rooted triplets in $O(n^3)$ time. This algorithm is used by our main algorithm *FastGalledNetwork* in section 3 to construct a galled network consistent with a given dense set \mathcal{T} of rooted triplets, if one exists, in optimal $O(n^3)$ time. In section 4, we prove that the problem becomes NP-hard if we remove the requirement that \mathcal{T} forms a dense set. Next, in section 5.1, we show that for every positive integer n , there exists some set \mathcal{T} of rooted triplets with $|\Lambda(\mathcal{T})| = n$ such that any galled network can be consistent with at most $0.4883 \cdot |\mathcal{T}|$ of the rooted triplets in \mathcal{T} . On the other hand, we give an $O(n \cdot |\mathcal{T}|^3)$ -time algorithm in section 5.2 that constructs a galled network guaranteed to be consistent with at least a factor of $\frac{5}{12}$ (> 0.4166) of the rooted triplets in \mathcal{T} for any input \mathcal{T} .

2. Constructing all simple phylogenetic networks when \mathcal{T} is dense. In this section, we describe an algorithm called *SimpleNetworks* for inferring all simple phylogenetic networks consistent with a given dense set \mathcal{T} of rooted triplets in $O(n^3)$ time, where $L = \Lambda(\mathcal{T})$ and $n = |L|$. This algorithm is later used by our main algorithm in section 3. Below, for any $L' \subseteq L$, $\mathcal{G}(L')$ denotes *the auxiliary graph for L'* (originally defined by Aho et al. [1]), which is the undirected graph with vertex set L' and edge set $E(L')$, where for each $(\{i, j\}, k) \in \mathcal{T} \mid L'$, the edge $\{i, j\}$ is included in $E(L')$.

SimpleNetworks assumes that $n \geq 3$, \mathcal{T} is dense, and $\mathcal{G}(L)$ is connected.

For any simple network N , define $A(N)$ and $B(N)$ to be the sets of leaves attached to the two clipped merge paths in N , where we require without loss of generality that $A(N)$ is nonempty. If both $A(N)$ and $B(N)$ are nonempty, then N is called *nonskew*; if $A(N)$ is nonempty and $B(N)$ is empty, then N is called *skew*.

Algorithm *SimpleNetworks* is listed in Figure 2. It calls two procedures named *Non-SkewSimpleNetworks* and *SkewSimpleNetworks* that find all valid nonskew simple networks and all valid skew simple networks, respectively. Then it returns their union. In the next two subsections, we show how to implement each of these two procedures to run in $O(n^3)$ time. We thus obtain Theorem 1.

THEOREM 1. *The set of all simple networks consistent with a given dense set of rooted triplets and leaf-labeled by L can be constructed in $O(n^3)$ time.*

The next two lemmas are used in sections 2.1 and 2.2. A *caterpillar tree* is a rooted tree such that every internal node has at most one child which is not a leaf (see, e.g., [2]). Recall that for any simple network N , we denote the leaf attached to the hybrid node in N by $hl(N)$.

LEMMA 1. *Suppose N is a simple phylogenetic network that is consistent with a set \mathcal{T} of rooted triplets. Let N' be the graph obtained from N by deleting the root node of N , the hybrid node of N , and $hl(N)$ together with all their incident edges, and then, for every node with outdegree 1 and indegree less than 2, contracting its*

Algorithm *SimpleNetworks*

Input: A dense set \mathcal{T} of rooted triplets with a leaf set L such that $\mathcal{G}(L)$ consists of one connected component.

Output: The set of all simple phylogenetic networks with leaf set L which are consistent with \mathcal{T} .

- 1 Let $\mathcal{N}_1 = \text{Non-SkewSimpleNetworks}(\mathcal{T})$.
- 2 Let $\mathcal{N}_2 = \text{SkewSimpleNetworks}(\mathcal{T})$.
- 3 **return** $\mathcal{N}_1 \cup \mathcal{N}_2$.

End *SimpleNetworks*

FIG. 2. Constructing all simple phylogenetic networks.

outgoing edge. If N is nonskew, then N' consists of two binary caterpillar trees which are consistent with $\mathcal{T} \upharpoonright A(N)$ and $\mathcal{T} \upharpoonright B(N)$, respectively; if N is skew, then N' is a binary caterpillar tree which is consistent with $\mathcal{T} \upharpoonright A(N)$.

LEMMA 2. [15] Let \mathcal{T} be a dense set of rooted triplets and let L be the leaf set of \mathcal{T} . There is at most one rooted, unordered tree distinctly leaf-labeled by L which is consistent with \mathcal{T} . Furthermore, if such a tree exists, then it must be binary.

2.1. Constructing all nonskew simple phylogenetic networks. Let U be an undirected, connected graph. Any partition (X, Y, Z) of the vertex set of U is called a *nonskew leaf partition in U* if $|X| \geq 1$, $|Y| = 1$, $|Z| \geq 1$, and in U the following holds: (1) X and Z form two cliques, and (2) there is no edge between a vertex in X and a vertex in Z . Any two nonskew leaf partitions of the form (X, Y, Z) and (Z, Y, X) are considered to be equivalent.

LEMMA 3. Let \mathcal{T} be a dense set of rooted triplets and let L be the leaf set of \mathcal{T} . If N is a nonskew simple phylogenetic network with leaf set L that is consistent with \mathcal{T} , then $(A(N), hl(N), B(N))$ forms a nonskew leaf partition in $\mathcal{G}(L)$.

Proof. Consider any $a_i, a_j \in A(N)$ with $i \neq j$ and let b be an arbitrary element in $B(N)$. N is consistent with \mathcal{T} , implying that $(\{a_i, b\}, a_j) \notin \mathcal{T}$ and $(\{a_j, b\}, a_i) \notin \mathcal{T}$. Since \mathcal{T} is dense, $(\{a_i, a_j\}, b)$ must belong to \mathcal{T} ; i.e., there is an edge (a_i, a_j) in $\mathcal{G}(L)$. Hence, $A(N)$ forms a clique in $\mathcal{G}(L)$. In the same way, $B(N)$ forms a clique in $\mathcal{G}(L)$. Moreover, \mathcal{T} cannot contain any rooted triplet of the form $(\{a, b\}, x)$ where $a \in A(N)$, $b \in B(N)$, $x \in L$, and thus there are no edges in $\mathcal{G}(L)$ between leaves in $A(N)$ and leaves in $B(N)$. However, $\mathcal{G}(L)$ is connected, which means that there must be at least one edge from $A(N)$ to the leaf $hl(N)$ and at least one edge from $B(N)$ to $hl(N)$. By definition, $(A(N), hl(N), B(N))$ forms a nonskew leaf partition in $\mathcal{G}(L)$. \square

By Lemmas 1 and 3, if N is a nonskew simple network with leaf set L that is consistent with \mathcal{T} , then $(A(N), hl(N), B(N))$ forms a nonskew leaf partition in $\mathcal{G}(L)$ and $\mathcal{T} \upharpoonright A(N)$ and $\mathcal{T} \upharpoonright B(N)$ are consistent with two binary caterpillar trees. Algorithm *Non-SkewSimpleNetworks*, shown in Figure 3, uses these implications to efficiently construct all nonskew simple networks with leaf set L that are consistent with \mathcal{T} . The algorithm enumerates all nonskew leaf partitions in $\mathcal{G}(L)$, and for each such leaf partition P , tries to build binary caterpillar trees for subsets of L induced by P (Lemma 2 ensures that for any dense subset \mathcal{T}' of \mathcal{T} , if \mathcal{T}' is consistent with a caterpillar tree, then it is uniquely determined, and so the algorithm of Aho et al. [1] can find it) and if successful, then combines the caterpillar trees in accordance with Lemma 1 to obtain all possible valid simple networks. Lemmas 1 and 3 guarantee that this approach will discover every valid simple network. However, it may also yield some simple networks which are not consistent with \mathcal{T} ; hence, before including any

<p>Algorithm <i>Non-SkewSimpleNetworks</i></p> <p>Input: A dense set \mathcal{T} of rooted triplets with a leaf set L such that $\mathcal{G}(L)$ consists of one connected component.</p> <p>Output: The set of all nonskew simple phylogenetic networks with leaf set L which are consistent with \mathcal{T}.</p> <ol style="list-style-type: none"> 1 Set $\mathcal{N}_1 = \emptyset$. 2 Construct $\mathcal{G}(L)$ and compute all nonskew leaf partitions in $\mathcal{G}(L)$. 3 for every nonskew leaf partition (X, Y, Z) in $\mathcal{G}(L)$ do 3.1 Let $T_X = \text{BuildTree}(\mathcal{T} X)$ and $T_Z = \text{BuildTree}(\mathcal{T} Z)$. 3.2 if T_X and T_Z are binary caterpillar trees then Make the roots of T_X and T_Z children of a new root node, create a new hybrid node H with a child labeled by the leaf in Y, and construct at most four nonskew simple networks by attaching H to one of T_X's bottommost leaves' parent edges and one of T_Z's bottommost leaves' parent edges in all possible ways. For each obtained network N, if N is consistent with \mathcal{T} then let $\mathcal{N}_1 = \mathcal{N}_1 \cup \{N\}$. endfor 4 return \mathcal{N}_1. <p>End <i>Non-SkewSimpleNetworks</i></p>
--

FIG. 3. Constructing all nonskew simple phylogenetic networks.

constructed network N in the final solution set \mathcal{N}_1 , *Non-SkewSimpleNetworks* verifies if N is consistent with \mathcal{T} .

For any $L' \subseteq L$ with $|L'| \geq 3$, $\text{BuildTree}(\mathcal{T} | L')$ refers to the fast implementation of the algorithm of Aho et al. applied to $\mathcal{T} | L'$ (we may assume it returns *null* if it fails). For $|L'| < 3$, the set $\mathcal{T} | L'$ is empty and we simply let $\text{BuildTree}(\mathcal{T} | L')$ return a tree with the one or two leaves in L' . The running time of $\text{BuildTree}(\mathcal{T} | L')$ is $\min\{O(|\mathcal{T}| \cdot \log^2 n), O(|\mathcal{T}| + n^2 \log n)\}$ (see section 1.3).

We now derive an upper bound on the running time of *Non-SkewSimpleNetworks*.

LEMMA 4. *For any undirected, connected graph U with n vertices, all nonskew leaf partitions in U can be computed in $O(n^3)$ time.*

Proof. To find all nonskew leaf partitions in U , test each of the n vertices to see if its removal divides U into two disjoint, nonempty cliques. Each test can be done in $O(n^2)$ time by depth-first search; thus this takes a total of $O(n^3)$ time. \square

LEMMA 5. *Any undirected, connected graph U has at most two nonskew leaf partitions.*

Proof. First observe that for any two different nonskew leaf partitions $(X, \{h\}, Z)$ and $(X', \{h'\}, Z')$ in U , we have $h \neq h'$. Moreover, h and h' are neighbors in U (otherwise, for any two neighbors x', z' of h' such that $x' \in X'$ and $z' \in Z'$, we have $x' \neq h$ and $z' \neq h$, and then all of h', x', z' must belong to one of X and Z while there is no edge between x' and z' , which is a contradiction).

Now, suppose $(X, \{h\}, Z)$ is a nonskew leaf partition in U and consider any other nonskew leaf partition $(X', \{h'\}, Z')$ in U . Either $h' \in X$ or $h' \in Z$. If $h' \in X$, then h can have no neighbors in X other than h' (otherwise, there would be an edge between X' and Z') and, furthermore, U cannot have any nonskew leaf partition of the form $(X'', \{h''\}, Z'')$ where $h'' \in Z$ because h' and h'' are not neighbors. This also holds in the case $h' \in Z$. This proves that U has at most two nonskew leaf partitions. \square

Algorithm *SkewSimpleNetworks*

Input: A dense set \mathcal{T} of rooted triplets with a leaf set L such that $\mathcal{G}(L)$ consists of one connected component.

Output: The set of all skew simple phylogenetic networks with leaf set L which are consistent with \mathcal{T} .

- 1 Set $\mathcal{N}_2 = \emptyset$.
- 2 Construct \mathcal{D} .
- 3 **for** every $x \in L$ **do**
- 3.1 Let $Q = \text{BuildCaterpillar}(\mathcal{D} | L')$, where $L' = L \setminus \{x\}$.
- 3.2 **if** $Q \neq \text{null}$ **then**
- Make the root of Q a child of a new root node r , create a new hybrid node H with a child labeled by x , add an edge from r to H , and construct two skew simple networks by attaching H to each one of Q 's two bottommost edges.
- For each obtained network N , if N is consistent with all rooted triplets in \mathcal{T} involving x then let $\mathcal{N}_2 = \mathcal{N}_2 \cup \{N\}$.
- endif**
- 4 **return** \mathcal{N}_2 .

End *SkewSimpleNetworks*

FIG. 4. Constructing all skew simple phylogenetic networks.

Next, if N is any galled network with n leaves, then the total number of nodes in N is $O(n)$ by Lemma 3 in [4]. By traversing the $O(n)$ nodes in N in a bottom-up order while keeping track of each node's $O(n)$ descendants and updating a table containing all $O(n^2)$ node pairs' lowest common ancestors, we obtain the following.

LEMMA 6. *Let N be a galled network with n leaves. After $O(n^2)$ time preprocessing, we can check if any given rooted triplet is consistent with N in $O(1)$ time.*

THEOREM 2. *The time complexity of Algorithm Non-SkewSimpleNetworks is $O(n^3)$.*

Proof. Steps 1 and 4 require $O(1)$ time. Step 2 can be performed in $O(n^3)$ time by a single scan of \mathcal{T} and by applying Lemma 4. Moreover, $\mathcal{G}(L)$ has at most two nonskew leaf partitions according to Lemma 5. Therefore, steps 3.1 and 3.2 are carried out at most two times each. Step 3.1 takes $O(|\mathcal{T}| + n^2 \log n) = O(n^3)$ time with the fast implementation of *BuildTree* by Henzinger, King, and Warnow [10]. Every time step 3.2 is performed, the algorithm constructs at most four nonskew simple networks and tests each of them for inclusion in \mathcal{N} , which after $O(n^2)$ time preprocessing takes $O(n^3)$ time using Lemma 6 since $|\mathcal{T}| = O(n^3)$. Hence, the total running time is $O(n^3)$. \square

2.2. Constructing all skew simple phylogenetic networks. To obtain all skew simple networks with leaf set L consistent with \mathcal{T} , Algorithm *SkewSimpleNetworks* in Figure 4 tries all ways to remove one leaf x from L and construct a binary caterpillar tree consistent with all rooted triplets not involving x using a procedure named *BuildCaterpillar*. For each such caterpillar Q , it forms two candidate skew simple networks by letting the root of Q be a child of a new split node with a hybrid node H such that H has a child labeled by x and H is attached to one of Q 's two bottommost edges. (By Lemma 1, every skew simple network with leaf set L that is consistent with \mathcal{T} must have this structure.) Then, each candidate skew simple network is checked to see if it is consistent with all rooted triplets in \mathcal{T} involving x (by the above, it is always consistent with the rest); if yes, then it is included in the solution set \mathcal{N}_2 .

The procedure *BuildCaterpillar* uses a graph \mathcal{D} , defined as follows. Given a set \mathcal{T} of rooted triplets with leaf set L , let \mathcal{D} be the directed graph with vertex set L such that there is a directed edge (x, y) if and only if \mathcal{T} contains at least one rooted triplet of the form $(\{y, z\}, x)$, where $z \in L$. For any $L' \subseteq L$, let $\mathcal{D}|L'$ be the subgraph of \mathcal{D} in which all vertices not in L' and their incident edges have been deleted. $\mathcal{D}|L'$ is acyclic if and only if there exists a binary caterpillar tree consistent with $\mathcal{T}|L'$.

For any $L' \subseteq L$, *BuildCaterpillar*($\mathcal{D}|L'$) returns a binary caterpillar tree with leaf set L' which is consistent with $\mathcal{T}|L'$ if such a tree exists, and *null* otherwise, by the following method. If $\mathcal{D}|L'$ has a cycle, then return *null*. Else, do a topological sort of $\mathcal{D}|L'$ to find a linear ordering \mathcal{O} of L' and return a binary caterpillar tree whose leaves are labeled in order of increasing distance from the root according to \mathcal{O} . Since \mathcal{T} is dense, \mathcal{O} is uniquely determined except for its last two elements which may be interchanged arbitrarily.

THEOREM 3. *The time complexity of Algorithm *SkewSimpleNetworks* is $O(n^3)$.*

Proof. Steps 1 and 4 require $O(1)$ time, and step 2 can be performed in $O(n^3)$ time by a single scan of \mathcal{T} . Steps 3.1 and 3.2 are carried out n times. Each call to *BuildCaterpillar* in step 3.1 takes $O(n^2)$ time since a topological sort can be done in $O(n^2)$ time. In step 3.2, to construct two networks and test them against the $O(n^2)$ rooted triplets involving x takes $O(n^2)$ time by Lemma 6. Hence, the total running time is $O(n^3)$. \square

3. An exact algorithm for inferring a galled phylogenetic network from a dense set of rooted triplets with optimal running time. Here, we present our algorithm *FastGalledNetwork* for constructing a galled network consistent with a given dense set \mathcal{T} of rooted triplets if such a network exists. Its running time is $O(n^3)$, where $n = |L|$ and L denotes the leaf set of \mathcal{T} , which is optimal since the size of the input is $\Theta(n^3)$ when \mathcal{T} is dense.

In section 3.1, we give an algorithm named *ComputeSNTree* which computes the so-called *SN-tree* for \mathcal{T} in $O(n^3)$ time. Then, in section 3.2, we describe *FastGalledNetwork*. It uses *ComputeSNTree* as well as *SimpleNetworks* from section 2 to construct a galled network consistent with \mathcal{T} (if one exists) from the *SN-tree* for \mathcal{T} . In sections 3.1 and 3.2 below, we assume \mathcal{T} is dense.

3.1. Computing the *SN-tree*. For any $X \subseteq L$, the set $SN(X)$ is defined recursively as $SN(X \cup \{c\})$ if there exist some $x, x' \in X$ and $c \in L \setminus X$ such that $(\{x, c\}, x') \in \mathcal{T}$, and as X otherwise. *SN-sets* were introduced in [15]. Intuitively, each *SN-set* is a subset of L which will form the leaf set of one subnetwork in the final solution. The *SN-sets* satisfy the following important property.

LEMMA 7 (see [15]). *If \mathcal{T} is dense, then for any $A, B \subseteq L$, $SN(A) \cap SN(B)$ equals \emptyset , $SN(A)$, or $SN(B)$.*

Reference [15] showed how to compute $SN(\{a, b\})$ for any $a, b \in L$ in $O(n^3)$ time; that approach therefore takes $O(n^5)$ time to compute $SN(\{a, b\})$ for all $a, b \in L$. This section presents a faster method for implicitly computing all *SN-sets* of this form when \mathcal{T} is dense, which requires only $O(n^3)$ time. The algorithm (*ComputeSNTree*) is listed in Figure 5. Given a dense \mathcal{T} , it builds a rooted tree called *the SN-tree for \mathcal{T}* which encodes all *SN-sets* so that $SN(X)$ for any $X \subseteq L$ can be retrieved efficiently.

In the first step, *ComputeSNTree* constructs a directed graph $G_{\mathcal{T}}$ with vertex set $V(G_{\mathcal{T}})$ and edge set $E(G_{\mathcal{T}})$. $V(G_{\mathcal{T}})$ is defined as $\{v_{\{a,b\}} \mid a, b \in L\}$, where $v_{\{a,a\}}$ for any $a \in L$ is denoted by $v_{\{a\}}$ for short, and $E(G_{\mathcal{T}})$ is $\{(v_{\{a,c\}}, v_{\{a,b\}}), (v_{\{a,c\}}, v_{\{b,c\}}), (v_{\{b,c\}}, v_{\{a,b\}}), (v_{\{b,c\}}, v_{\{a,c\}}) \mid (\{a, b\}, c) \in \mathcal{T}\} \cup \{(v_{\{a,b\}}, v_{\{a\}}), (v_{\{a,b\}}, v_{\{b\}}) \mid a, b \in L\}$.

Algorithm *ComputeSNTree*
Input: A dense set \mathcal{T} of rooted triplets with a leaf set L .
Output: The SN -tree $R_{\mathcal{T}}$ for \mathcal{T} .

- 1 Construct the directed graph $G_{\mathcal{T}}$.
- 2 Compute the set \mathcal{C} of strongly connected components of $G_{\mathcal{T}}$ and then construct $G'_{\mathcal{T}}$ for \mathcal{T} .
- 3 Construct the SN -tree $R_{\mathcal{T}}$ and **return** $R_{\mathcal{T}}$.

End *ComputeSNTree*

FIG. 5. Computing the SN -tree for a dense set \mathcal{T} .

L }. Note that $|V(G_{\mathcal{T}})| = O(n^2)$ and $|E(G_{\mathcal{T}})| = O(n^3)$. Before describing the remaining steps of *ComputeSNTree*, we first investigate the structure of $G_{\mathcal{T}}$ and the relationship between $G_{\mathcal{T}}$ and the SN -sets of the form $SN(\{a, b\})$.

LEMMA 8. For every $a, b, y, z \in L$, if $G_{\mathcal{T}}$ contains a path from $v_{\{a,b\}}$ to $v_{\{a,y\}}$ and a path from $v_{\{a,b\}}$ to $v_{\{a,z\}}$, then $G_{\mathcal{T}}$ has a path from $v_{\{a,b\}}$ to $v_{\{y,z\}}$.

Proof. If $|\{a, y, z\}| < 3$, then the lemma follows from the construction of $G_{\mathcal{T}}$. Otherwise, since \mathcal{T} is dense, we have one of the following cases for $\{a, y, z\}$:

- Case (1): $(\{y, z\}, a) \in \mathcal{T}$ or $(\{a, z\}, y) \in \mathcal{T}$. Then $(v_{\{a,y\}}, v_{\{y,z\}}) \in E(G_{\mathcal{T}})$, and thus there is a path from $v_{\{a,b\}}$ to $v_{\{a,y\}}$ and then to $v_{\{y,z\}}$.
- Case (2): $(\{a, y\}, z) \in \mathcal{T}$. Then $(v_{\{a,z\}}, v_{\{y,z\}}) \in E(G_{\mathcal{T}})$, and thus there is a path from $v_{\{a,b\}}$ to $v_{\{a,z\}}$ and then to $v_{\{y,z\}}$. \square

LEMMA 9. For every $a, b, c \in L$, if $c \in SN(\{a, b\})$, then there exists a directed path from $v_{\{a,b\}}$ to $v_{\{a,c\}}$ in $G_{\mathcal{T}}$.

Proof. Define $SN_0(\{a, b\}) = \{a, b\}$ and for $\ell = 1, 2, \dots, n$, define $SN_{\ell} = \{x \in L \mid (\{y, x\}, z) \in \mathcal{T} \text{ for some } y, z \in SN_{\ell-1}(\{a, b\})\}$. Note that $SN(\{a, b\}) = \bigcup_{i=0}^n SN_i(\{a, b\})$. We prove by induction that the following statement $P(\ell)$ is true for $\ell \in \{0, 1, 2, \dots, n\}$.

$P(\ell)$: For every $x \in SN_{\ell}(\{a, b\})$, there exists a path from $v_{\{a,b\}}$ to $v_{\{a,x\}}$ in $G_{\mathcal{T}}$.

When $\ell = 0$ (the base case), the statement follows trivially by the construction of $G_{\mathcal{T}}$.

Next, when $\ell > 0$, suppose the statement $P(\ell - 1)$ is true; i.e., for every $w \in SN_{\ell-1}(\{a, b\})$, there exists a path from $v_{\{a,b\}}$ to $v_{\{a,w\}}$ in $G_{\mathcal{T}}$. Consider any $x \in SN_{\ell}(\{a, b\})$. By the definition of SN_{ℓ} , there exist $y, z \in SN_{\ell-1}(\{a, b\})$ such that $(\{y, x\}, z) \in \mathcal{T}$. Then $P(\ell - 1)$ implies that there is a path from $v_{\{a,b\}}$ to $v_{\{a,y\}}$ and a path from $v_{\{a,b\}}$ to $v_{\{a,z\}}$, which means there exists a path from $v_{\{a,b\}}$ to $v_{\{y,z\}}$ according to Lemma 8. Moreover, $(v_{\{y,z\}}, v_{\{x,y\}}) \in E(G_{\mathcal{T}})$ because $(\{y, x\}, z) \in \mathcal{T}$. Now, if $x = a$ or $y = a$, then $P(\ell)$ follows directly; therefore assume $x \neq a$ and $y \neq a$. Since \mathcal{T} is dense, for the set $\{a, x, y\}$, there are two cases:

- Case (1): $(\{a, y\}, x) \in \mathcal{T}$ or $(\{a, x\}, y) \in \mathcal{T}$. Then $(v_{\{x,y\}}, v_{\{a,x\}}) \in E(G_{\mathcal{T}})$.
- Case (2): $(\{x, y\}, a) \in \mathcal{T}$. Then $(v_{\{a,y\}}, v_{\{a,x\}}) \in E(G_{\mathcal{T}})$.

In both cases, there exists a path from $v_{\{a,b\}}$ to $v_{\{a,x\}}$, and thus $P(\ell)$ holds.

By induction, $P(\ell)$ is true for every $\ell \in \{0, 1, 2, \dots, n\}$. Since c belongs to at least one set SN_{ℓ} , the lemma follows. \square

LEMMA 10. For every $a, b, c, d \in L$, if there is a directed path from $v_{\{a,b\}}$ to $v_{\{c,d\}}$ in $G_{\mathcal{T}}$, then $SN(\{c, d\}) \subseteq SN(\{a, b\})$.

Proof. For any $e \in SN(\{c, d\})$, there is a directed path from $v_{\{c,d\}}$ to $v_{\{c,e\}}$ by Lemma 9. Since $E(G_{\mathcal{T}})$ contains the directed edge $(v_{\{c,e\}}, v_{\{e\}})$, and since there is a directed path from $v_{\{a,b\}}$ to $v_{\{c,d\}}$, this means there is a directed path from $v_{\{a,b\}}$ to

$v_{\{e\}}$. Without loss of generality, let the path be $v_{\{x_0,x_1\}}, v_{\{x_1,x_2\}}, v_{\{x_2,x_3\}}, \dots, v_{\{x_{p-1},x_p\}}, v_{\{x_p\}}$, where $\{x_0, x_1\} = \{a, b\}$, and $x_p = e$. Then, based on the rooted triplets in \mathcal{T} for the sets $\{x_0, x_1, x_2\}, \{x_1, x_2, x_3\}, \dots, \{x_{p-2}, x_{p-1}, x_p\}$, we can deduce that $x_p \in SN(\{a, b\})$. Thus, $e \in SN(\{a, b\})$, so we have just shown that $SN(\{c, d\}) \subseteq SN(\{a, b\})$. \square

COROLLARY 1. *For any two nodes $v_{\{a,b\}}$ and $v_{\{c,d\}}$ on a directed cycle in $G_{\mathcal{T}}$, $SN(\{a, b\}) = SN(\{c, d\})$.*

By the above, computing $SN(\{a, b\})$ for any $a, b \in L$ is equivalent to finding all nodes of the form $v_{\{c\}}$ reachable from $v_{\{a,b\}}$. Let the set of all strongly connected components of $G_{\mathcal{T}}$ be $\mathcal{C} = \{C_1, C_2, \dots, C_m\}$. By Corollary 1, $SN(\{a, b\}) = SN(\{c, d\})$ if $v_{\{a,b\}}$ and $v_{\{c,d\}}$ are in the same C_i . So, we define $SN(C_i)$ as $SN(\{a, b\})$ for any $v_{\{a,b\}} \in C_i$. The set \mathcal{C} has the following properties.

LEMMA 11. *For every $c \in L$, $\{v_{\{c\}}\} \in \mathcal{C}$. Moreover, for every $i \neq j$, $SN(C_i) \neq SN(C_j)$.*

Proof. Since $v_{\{c\}}$ has no outgoing edge, $\{v_{\{c\}}\}$ is a strongly connected component in $G_{\mathcal{T}}$ and therefore belongs to \mathcal{C} .

To prove the second statement, suppose for the sake of contradiction that there exist strongly connected components C_i, C_j with $i \neq j$ such that $SN(C_i) = SN(C_j)$. Take any $v_{\{w,x\}} \in C_i$ and $v_{\{y,z\}} \in C_j$. Since $y, z \in SN(C_j) = SN(C_i) = SN(\{w, x\})$, it follows from Lemma 9 that $G_{\mathcal{T}}$ has a path from $v_{\{w,x\}}$ to $v_{\{w,y\}}$ and a path from $v_{\{w,x\}}$ to $v_{\{w,z\}}$, and hence a path from $v_{\{w,x\}}$ to $v_{\{y,z\}}$ by Lemma 8. Symmetrically, $G_{\mathcal{T}}$ contains a path from $v_{\{y,z\}}$ to $v_{\{w,x\}}$. But then $C_i \cup C_j$ must be a strongly connected component of $G_{\mathcal{T}}$, and we have arrived at a contradiction. \square

In the second step of *ComputeSNTree*, we compute \mathcal{C} and then let $G'_{\mathcal{T}}$ be the directed graph with vertex set $V(G'_{\mathcal{T}}) = \mathcal{C}$ and edge set $E(G'_{\mathcal{T}}) = \{(C_i, C_j) \mid \text{there exists some } (v_{\{w,x\}}, v_{\{y,z\}}) \in E(G_{\mathcal{T}}) \text{ where } v_{\{w,x\}} \in C_i \text{ and } v_{\{y,z\}} \in C_j\}$. Note that $G'_{\mathcal{T}}$ is a directed acyclic graph. From $G'_{\mathcal{T}}$, construct a graph $R_{\mathcal{T}}$ with $V(R_{\mathcal{T}}) = \mathcal{C}$ and $E(R_{\mathcal{T}}) = \{(C_i, C_j) \in E(G'_{\mathcal{T}}) \mid \text{there exists no path of length at least 2 from } C_i \text{ to } C_j \text{ in } G'_{\mathcal{T}}\}$. Finally, return $R_{\mathcal{T}}$. The next two lemmas show that $R_{\mathcal{T}}$ is indeed a tree.

LEMMA 12. *There is only one node in $G'_{\mathcal{T}}$ with indegree 0.*

Proof. Suppose $G'_{\mathcal{T}}$ has two different nodes r, s with indegree 0. Denote the two strongly connected components in $G_{\mathcal{T}}$ which correspond to r and s in $R_{\mathcal{T}}$ by C_r and C_s , respectively. Let $v_{\{a,b\}}$ be any node in C_r and let $v_{\{c,d\}}$ be any node in C_s . Clearly, $a \neq b$ and $c \neq d$ since any node of the form $v_{\{a\}}$ belongs to a strongly connected component consisting only of $v_{\{a\}}$ and therefore cannot have indegree 0 by the construction of $G_{\mathcal{T}}$. Consider the three possible rooted triplets with leaf set $\{a, b, c\}$ (of which at least one belongs to \mathcal{T} since \mathcal{T} is dense). By the definition of $E(G_{\mathcal{T}})$, there will always be at least one edge ending at $v_{\{a,b\}}$. Since r has indegree 0 in $G'_{\mathcal{T}}$, this implies that (1) at least one of $v_{\{a,c\}}$ and $v_{\{b,c\}}$ must be in C_r . In the same way, we see that (2) at least one of $v_{\{a,d\}}$ and $v_{\{b,d\}}$ is in C_r ; (3) at least one of $v_{\{a,c\}}$ and $v_{\{a,d\}}$ is in C_s ; and (4) at least one of $v_{\{b,c\}}$ and $v_{\{b,d\}}$ is in C_s .

Assume without loss of generality that $v_{\{a,c\}} \in C_r$. Then (3) yields $v_{\{a,d\}} \in C_s$, and thus we have $v_{\{b,d\}} \in C_r$ by (2), and then $v_{\{b,c\}} \in C_s$ by (4). There are three cases:

- Case (1): $(\{a, b\}, c) \in T$. Then the two edges $(v_{\{a,c\}}, v_{\{b,c\}})$ and $(v_{\{b,c\}}, v_{\{a,c\}})$ in $E(G_{\mathcal{T}})$ imply that C_r is reachable from C_s in $G'_{\mathcal{T}}$ and vice versa.
- Case (2): $(\{a, c\}, b) \in T$. Then $(v_{\{a,b\}}, v_{\{b,c\}}), (v_{\{b,c\}}, v_{\{a,b\}}) \in E(G_{\mathcal{T}})$ imply that C_r is reachable from C_s in $G'_{\mathcal{T}}$ and vice versa.
- Case (3): $(\{b, c\}, a) \in T$. Then $(v_{\{a,b\}}, v_{\{b,c\}}) \in E(G_{\mathcal{T}})$, and thus C_s is reachable from C_r in $G'_{\mathcal{T}}$. Next, by considering all possible rooted triplets on

$\{a, c, d\}$, we see that $(v_{\{c,d\}}, v_{\{a,c\}}) \in E(G_{\mathcal{T}})$; $(v_{\{a,c\}}, v_{\{c,d\}}), (v_{\{c,d\}}, v_{\{a,c\}}) \in E(G_{\mathcal{T}})$; or $(v_{\{a,c\}}, v_{\{a,d\}}), (v_{\{a,d\}}, v_{\{a,c\}}) \in E(G_{\mathcal{T}})$, which means that C_r is always reachable from C_s in $G'_{\mathcal{T}}$.

Every case contradicts that C_r and C_s are disjoint strongly connected components. Hence, $G'_{\mathcal{T}}$ can only have one node with indegree 0. \square

LEMMA 13. $R_{\mathcal{T}}$ is a tree with no nodes having outdegree 1. Its set of leaves is $\{\{v_{\{c\}}\} \mid c \in L\}$.

Proof. From Lemma 12 and by the construction of $R_{\mathcal{T}}$, it follows that $R_{\mathcal{T}}$ has only one node with indegree 0, i.e., only one root node.

Now, suppose $R_{\mathcal{T}}$ is not a tree. Then there exists a node C_j in $R_{\mathcal{T}}$ with at least two parents, say C_i and $C_{i'}$. By definition, $(C_i, C_j), (C_{i'}, C_j) \in E(G'_{\mathcal{T}})$ and by Lemma 10, $SN(C_j) \subseteq SN(C_i)$ as well as $SN(C_j) \subseteq SN(C_{i'})$. Next, by Lemmas 7 and 11, we have either $SN(C_i) \subsetneq SN(C_{i'})$ or $SN(C_{i'}) \subsetneq SN(C_i)$. Without loss of generality, assume $SN(C_i) \subsetneq SN(C_{i'})$. Then, by Lemmas 8 and 9, we have a path in $G'_{\mathcal{T}}$ from $C_{i'}$ to C_i and then to C_j . But this implies that $(C_{i'}, C_j) \notin E(R_{\mathcal{T}})$, which is a contradiction. Thus, $R_{\mathcal{T}}$ must be a tree.

Next, suppose some node C_i in $R_{\mathcal{T}}$ has a single child C_j . By Lemma 10, $SN(C_j) \subseteq SN(C_i)$. Observe that for any $c \in SN(C_i)$, there is a path from C_i to $v_{\{c\}}$ in $G'_{\mathcal{T}}$ according to Lemma 9, and since this path passes through C_j , we also have $c \in SN(C_j)$ by Lemma 10. This means that $SN(C_i) \subseteq SN(C_j)$, giving us $SN(C_i) = SN(C_j)$. But this contradicts Lemma 11. Thus, $R_{\mathcal{T}}$ has no nodes with outdegree 1.

Finally, for every $c \in L$, $\{v_{\{c\}}\}$ is of outdegree 0 in $G'_{\mathcal{T}}$ and is therefore a leaf in $R_{\mathcal{T}}$. The lemma follows. \square

COROLLARY 2. $|\mathcal{C}| = O(n)$.

Proof. By the definition of $R_{\mathcal{T}}$, we have $V(R_{\mathcal{T}}) = \mathcal{C}$. Lemma 13 states that $R_{\mathcal{T}}$ is a tree with n leaves and no nodes with outdegree 1. Hence, $|\mathcal{C}| = |V(R_{\mathcal{T}})| = O(n)$. \square

In the rest of the paper, $R_{\mathcal{T}}$ is called the *SN-tree for \mathcal{T}* . This is because $SN(\{a, b\})$ for any $a, b \in L$ can be obtained from $R_{\mathcal{T}}$ using the following theorem.

THEOREM 4. Given any $a, b \in L$, let u be the lowest common ancestor of $v_{\{a\}}$ and $v_{\{b\}}$ in $R_{\mathcal{T}}$. Then, $SN(\{a, b\}) = \{c \in L \mid v_{\{c\}} \text{ is a descendant of } u \text{ in } R_{\mathcal{T}}\}$.

Proof. We first prove that for any $c \in L$, if $v_{\{c\}}$ is a descendant of the lowest common ancestor u of $v_{\{a\}}$ and $v_{\{b\}}$ in $R_{\mathcal{T}}$, then $c \in SN(\{a, b\})$. Let $v_{\{y,z\}}$ be any node in the strongly connected component in $G_{\mathcal{T}}$ which corresponds to u in $R_{\mathcal{T}}$. Then $a \in SN(\{y, z\})$ by Lemma 10. Since also $a \in SN(\{a, b\})$, Lemma 7 implies that either (1) $SN(\{a, b\}) \subsetneq SN(\{y, z\})$ or (2) $SN(\{y, z\}) \subseteq SN(\{a, b\})$. If (1) holds, then there is a path in $G_{\mathcal{T}}$ from $v_{\{y,z\}}$ to $v_{\{a,b\}}$ by Lemma 9, but then u cannot be the lowest common ancestor of $v_{\{a\}}$ and $v_{\{b\}}$ in $R_{\mathcal{T}}$, which is a contradiction. Thus, (2) must hold, which means that $y, z \in SN(\{a, b\})$, so there is a path in $G_{\mathcal{T}}$ from $v_{\{a,b\}}$ to $v_{\{y,z\}}$ as can be seen by applying Lemma 9 two times and then Lemma 8. Now, since $v_{\{c\}}$ is a descendant of u in $R_{\mathcal{T}}$, there is a path in $G_{\mathcal{T}}$ from $v_{\{y,z\}}$ to $v_{\{c\}}$. This shows that $v_{\{c\}}$ is reachable from $v_{\{a,b\}}$ in $G_{\mathcal{T}}$, i.e., that $c \in SN(\{a, b\})$ by Lemma 10.

Next, we prove that if $c \in SN(\{a, b\})$, then $v_{\{c\}}$ is a descendant of u in $R_{\mathcal{T}}$. Take any $c \in SN(\{a, b\})$, and suppose that $v_{\{c\}}$ is not a descendant of u in $R_{\mathcal{T}}$. Then $v_{\{b\}}$ is a descendant of the lowest common ancestor u' of $v_{\{a\}}$ and $v_{\{c\}}$ in $R_{\mathcal{T}}$, so $b \in SN(\{a, c\})$ by the preceding paragraph and, similarly, $a \in SN(\{b, c\})$. But then, $SN(\{a, b\}) = SN(\{a, c\}) = SN(\{b, c\})$ by Lemma 9 and Corollary 1, which is impossible since this would imply that u and u' coincide. Therefore, $v_{\{c\}}$ must be a descendant of u in $R_{\mathcal{T}}$. \square

Thus, the *SN-tree* has the properties we want. The next theorem shows that the

SN -tree for \mathcal{T} can be constructed efficiently.

THEOREM 5. *The time complexity of Algorithm `ComputeSNTree` is $O(n^3)$.*

Proof. By scanning all rooted triplets in \mathcal{T} , we can construct $G_{\mathcal{T}}$ in step 1 in $O(|\mathcal{T}|) = O(n^3)$ time. For step 2, the time complexity is $O(|V(G_{\mathcal{T}})| + |E(G_{\mathcal{T}})|) = O(n^3)$. To build the SN -tree $R_{\mathcal{T}}$ in step 3, we need two substeps. First, for each node $C_i \in V(G'_{\mathcal{T}})$, we compute the set of all nodes that are reachable from C_i in $G'_{\mathcal{T}}$. By Corollary 2, $|V(G'_{\mathcal{T}})| = O(n)$ and thus $|E(G'_{\mathcal{T}})| = O(n^2)$, so this takes $O(n^2)$ time for each C_i using depth-first search, so $O(n^3)$ time in total. Next, we check, for each $(C_i, C_j) \in E(G'_{\mathcal{T}})$, if there is a path of length at least 2 from C_i to C_j . If the answer is no, then (C_i, C_j) is an edge in $R_{\mathcal{T}}$. Each such check can be performed in $O(n)$ time by asking if C_j is reachable from any of the children (except C_j itself) of C_i . Since there are $O(n^2)$ edges, we can check all edges in $O(n^3)$ time. Thus, the algorithm's total running time is $O(n^3)$. \square

3.2. Algorithm `FastGalledNetwork`. The main algorithm of this section, Algorithm `FastGalledNetwork`, is listed in Figure 6. The key observation is that a galled network consistent with \mathcal{T} (if one exists) can be obtained from the SN -tree for \mathcal{T} by replacing each internal node of degree 3 or higher with a subnetwork whose structure is inferred by Algorithm `SimpleNetworks`.

Recall that for any node u in a rooted, leaf-labeled tree R , $R[u]$ is the subtree of R rooted at u , and $\Lambda(R[u])$ denotes the set of leaves in $R[u]$. Below, $\mathcal{T} \upharpoonright u$ is shorthand for the set $\mathcal{T} \upharpoonright \Lambda(R[u])$.

In step 1, `FastGalledNetwork` computes the SN -tree R for \mathcal{T} using Algorithm `ComputeSNTree` from section 3.1. Then, in steps 2 and 3, it tries to construct a galled network N_u consistent with all rooted triplets in $\mathcal{T} \upharpoonright u$ for each node u in R in bottom-up order. If successful, it returns N_r , where r is the root of R (note that $\mathcal{T} = \mathcal{T} \upharpoonright r$); otherwise, it returns *null*. To obtain N_u for any node u in R , `FastGalledNetwork` proceeds as follows. Let q be the degree of u and denote the children of u by $\{u_1, u_2, \dots, u_q\}$. If $q = 0$, then let N_u be a network consisting of one leaf, labeled by u . If $q = 2$, then form N_u by joining the roots of N_{u_1} and N_{u_2} to a new root node. Otherwise, $q \geq 3$ by Lemma 13. In this case, let $\alpha_1, \alpha_2, \dots, \alpha_q$ be q new symbols not in L , and define a function f as follows. For every $x \in \Lambda(R[u])$, let $f(x) = \alpha_i$, where $x \in \Lambda(R[u_i])$. Next, define \mathcal{T}' as the set $\{(\{f(x), f(y)\}, f(z)) : (\{x, y\}, z) \in (\mathcal{T} \upharpoonright u) \text{ and } f(x), f(y), f(z) \text{ all differ}\}$, and apply Algorithm `SimpleNetworks` from section 2 to \mathcal{T}' . If there is a simple phylogenetic network N' consistent with \mathcal{T}' , then replace each α_i in N' with N_{u_i} and let N_u be the resulting network; otherwise, terminate and output *null*.

The correctness of this method follows from the next two lemmas.

LEMMA 14. *For any node u in R , if $\mathcal{T} \upharpoonright u$ is consistent with a galled network with leaf set $\Lambda(R[u])$ and if $q \geq 3$, then there exists a simple network consistent with \mathcal{T}' .*

Proof. Let M be a galled network with leaf set $\Lambda(R[u])$ consistent with $\mathcal{T} \upharpoonright u$. First we show that if $q \geq 3$, then the root r of M must be a split node. Suppose r is not a split node and let A and B be the disjoint sets of leaves in the two subnetworks rooted at the children of r . For every child u_i of u , we have either $\Lambda(R[u_i]) \subseteq A$ or $\Lambda(R[u_i]) \subseteq B$ (otherwise, let a, b be two leaves in $\Lambda(R[u_i])$ such that $a \in A$ and $b \in B$; for each $x \in \Lambda(R[u]) \setminus \{a, b\}$, at least one of $(\{a, x\}, b)$ and $(\{b, x\}, a)$ belongs to $\mathcal{T} \upharpoonright u$ since \mathcal{T} is dense, so $x \in \Lambda(R[u_i])$, i.e., $\Lambda(R[u_i]) = \Lambda(R[u])$, which is not possible). Since $q \geq 3$, there exist i, j, k where i, j, k differ such that both $\Lambda(R[u_i])$ and $\Lambda(R[u_j])$ are subsets of one of A and B , and $\Lambda(R[u_k])$ is a subset of the other. Assume without loss of generality that $\Lambda(R[u_i]), \Lambda(R[u_j]) \subseteq A$ and $\Lambda(R[u_k]) \subseteq B$.

Algorithm *FastGalledNetwork*
Input: A dense set \mathcal{T} of rooted triplets with a leaf set L .
Output: A galled network consistent with \mathcal{T} , if one exists; otherwise, *null*.

- 1 Let $R = \text{ComputeSNTree}(\mathcal{T})$.
- 2 Define N_u for every leaf u in R to be a single node labeled by u .
- 3 **for** each nonleaf node u in R , in bottom-up order **do**
 /* Construct a galled network N_u for the set of leaves in $\Lambda(R[u])$. */
 3.1 Denote the set of children of u in R by $\{u_1, u_2, \dots, u_q\}$.
 3.2 If $q = 2$, let N_u be a network with a root node joined to N_{u_1} and N_{u_2} .
 3.3 Otherwise ($q \geq 3$), build \mathcal{T}' from $\mathcal{T} \upharpoonright u$, compute $\mathcal{N} = \text{SimpleNetworks}(\mathcal{T}')$, and check if \mathcal{N} is empty; if yes then **return** *null*, else select any $N' \in \mathcal{N}$ and form a network N_u by replacing each α_i in N' with N_{u_i} .
 endfor
- 4 **return** N_r , where r is the root of R .

End *FastGalledNetwork*

FIG. 6. Constructing a galled phylogenetic network consistent with a dense set \mathcal{T} of rooted triplets.

For any $x, y \in A$ and $b_k \in \Lambda(R[u_k])$, \mathcal{T} cannot contain $(\{x, b_k\}, y)$ or $(\{y, b_k\}, x)$, so $SN(\{a_i, a_j\})$ is a proper subset of $SN(\{a_i, b_k\})$ for every $a_i \in \Lambda(R[u_i])$, $a_j \in \Lambda(R[u_j])$. However, u is the lowest common ancestor in R of $\{a_i, a_j\}$ as well as of $\{a_i, b_k\}$, so $SN(\{a_i, a_j\}) = SN(\{a_i, b_k\})$ by Theorem 4, which is a contradiction. Hence, r is a split node.

Next, observe that each side network of M can contain leaves from only one $R[u_i]$. To see this, let $M[v]$ be a side network of M . For any i, j with $i \neq j$, if $M[v]$ contains a leaf $a \in \Lambda(R[u_i])$ and a leaf $b \in \Lambda(R[u_j])$, then since \mathcal{T} is dense, $\Lambda(R[u_i]) \subsetneq SN(\{a, b\})$ and $\Lambda(R[u_j]) \subsetneq SN(\{a, b\})$ by Lemma 7 while $SN(\{a, b\}) \neq \Lambda(R[u])$ (otherwise, $M[v]$ cannot be a side network of M), contradicting the maximality of $\Lambda(R[u_i])$ and $\Lambda(R[u_j])$ in $\Lambda(R[u])$.

By the preceding two paragraphs, the root of M is a split node of some hybrid node h , and each side network attached to a merge path of h contains leaves from only one $\Lambda(R[u_i])$. We now show that there exists a galled network M^* consistent with $\mathcal{T} \upharpoonright u$ such that for every i , all leaves in $\Lambda(R[u_i])$ belong to only one side network of M^* attached to a merge path of h . Suppose M has two side networks $M[v]$ and $M[w]$ attached to merge paths of h such that both $M[v]$ and $M[w]$ contain leaves from the same $\Lambda(R[u_i])$. $M[v]$ and $M[w]$ must be attached to the same merge path p of h (otherwise, $\Lambda(R[u_i]) = \Lambda(R[u])$, which is impossible), and, furthermore, all side networks attached to p between $M[v]$ and $M[w]$ contain leaves from $\Lambda(R[u_i])$ only. Thus, all side networks containing leaves from the same $\Lambda(R[u_i])$ are consecutively ordered along one merge path of h and can therefore be concatenated into one side network in such a way that all rooted triplets involving $\Lambda(R[u_i])$ are still consistent with it (note that \mathcal{T} does not contain any rooted triplet of the form $(\{a, x\}, b)$ where $a, b \in \Lambda(R[u_i])$ and x is located in a side tree of M below the side trees leaf-labeled by $\Lambda(R[u_i])$). Let M^* be the resulting galled network consistent with $\mathcal{T} \upharpoonright u$ such that each side network M_i^* attached to a merge path of h is bijectively leaf-labeled by one $\Lambda(R[u_i])$.

Finally, construct a simple phylogenetic network M' from M^* by replacing each M_i^* by a leaf labeled by α_i . M' is consistent with \mathcal{T}' , which can be seen as follows. Let t' be any rooted triplet in \mathcal{T}' and write $t' = (\{\alpha_i, \alpha_j\}, \alpha_k)$. Then there exists some rooted triplet $t = (\{x, y\}, z)$ in \mathcal{T} such that $x \in \Lambda(R[u_i])$, $y \in \Lambda(R[u_j])$, and

$z \in \Lambda(R[u_k])$, where i, j, k all differ. t is consistent with M^* , so t' is consistent with M' by the construction of M' . Hence, \mathcal{T}' and M' are consistent. \square

LEMMA 15. *Let u be any node in R and suppose each $\mathcal{T}|_{u_i}$ is consistent with a galled network N_{u_i} . If $q = 2$, then the galled network obtained by joining the roots of N_{u_1} and N_{u_2} to a new root node is consistent with $\mathcal{T}|_u$. If $q \geq 3$ and \mathcal{T}' is consistent with a simple network N' with leaf set $\{\alpha_1, \alpha_2, \dots, \alpha_q\}$, then the galled network N_u obtained from N' by replacing each α_i by N_{u_i} is consistent with $\mathcal{T}|_u$.*

Proof (analogous to the proof of Lemma 8 in [15]). Consider any rooted triplet t in $\mathcal{T}|_u$ and write $t = (\{x, y\}, z)$. If $x \in \Lambda(R[u_i])$, $y \in \Lambda(R[u_j])$, and $z \in \Lambda(R[u_k])$, where i, j, k all differ, then t is consistent with N_u (otherwise, $t' = (\{f(x), f(y)\}, f(z)) = (\{\alpha_i, \alpha_j\}, \alpha_k)$ cannot be consistent with N' which is a contradiction since $t' \in \mathcal{T}'$). If $x, y \in \Lambda(R[u_i])$ and $z \in \Lambda(R[u_j])$ with $i \neq j$, then t is consistent with N_u by the construction of N_u . The case $x, z \in \Lambda(R[u_i])$ and $y \in \Lambda(R[u_j])$ (or symmetrically, $y, z \in \Lambda(R[u_i])$ and $x \in \Lambda(R[u_j])$) with $i \neq j$ is not possible because then y would not belong to $SN(\{x, z\})$ by Theorem 4, contradicting that $y \in SN(\{x, z\})$ according to the definition of SN -sets. Finally, if x, y, z belong to the same $\Lambda(R[u_i])$, then t is consistent with N_{u_i} and therefore with N_u . In all possible cases, t is consistent with N_u . \square

We now analyze the running time of *FastGalledNetwork*.

THEOREM 6. *The time complexity of Algorithm *FastGalledNetwork* is $O(n^3)$.*

Proof. Step 1 takes $O(n^3)$ by Theorem 5. For every node u in R with $\deg(u) < 3$, N_u can be constructed in $O(1)$ time. The total time for constructing all networks N_u with $\deg(u) \geq 3$ is given by the total time needed to build all the \mathcal{T}' -sets plus the total time taken by all calls to *SimpleNetworks*; both of these are shown below to be $O(n^3)$. Thus, the theorem follows.

First note that to construct \mathcal{T}' for a node u , we need to consider only rooted triplets in \mathcal{T} whose three leaves belong to subtrees rooted at three different children of u . For this purpose, we may create a list $T(u)$ for each node u in R containing all rooted triplets in \mathcal{T} of the form $(\{x, y\}, z)$ such that u is the lowest common ancestor in R of x, y , and z . All the $T(u)$ -lists can be constructed using an additional $O(n^3)$ time after computing the SN -tree R in step 1 by doing a bottom-up traversal of R . Then, when constructing \mathcal{T}' in step 3.3, check each rooted triplet in $T(u)$ to see if its leaves belong to three different subtrees, and if so, update \mathcal{T}' accordingly. This way, each rooted triplet in \mathcal{T} is considered for one \mathcal{T}' -set only, so the total time required to build all \mathcal{T}' -sets is bounded by $O(n^3)$.

Next, note that each constructed \mathcal{T}' has $\deg(u)$ leaves. Running *SimpleNetworks* on \mathcal{T}' therefore takes $O((\deg(u))^3)$ time by Theorem 1. Summing over all nodes, the calls to *SimpleNetworks* take a total of $\sum_{u \in R} O((\deg(u))^3) = O((\sum_{u \in R} \deg(u))^3) = O(n^3)$ time. \square

Finally, we remark that *FastGalledNetwork* can be modified to return *all* galled networks consistent with \mathcal{T} by utilizing all simple networks computed in step 3.3. However, this may take exponential time.

4. NP-hardness of the nondense case. We now prove that the problem of inferring a galled phylogenetic network which is consistent with a given set of \mathcal{T} rooted triplets, if one exists, is NP-hard when \mathcal{T} is not required to be dense. Our proof consists of a polynomial-time reduction from the NP-complete problem Set Splitting (see, e.g., [5]) to the decision version of our problem. We use the same reduction to prove that the closely related problem of inferring a *simple* phylogenetic network which is consistent with a given (nondense) set of rooted triplets is also NP-hard.

Set Splitting. Given a set $S = \{s_1, s_2, \dots, s_n\}$ and a collection $\mathcal{C} = \{C_1, C_2, \dots, C_m\}$ of subsets of S , where $|C_j| = 3$ for every $C_j \in \mathcal{C}$, does (S, \mathcal{C}) have a set splitting; i.e., can S be partitioned into two disjoint subsets S_1, S_2 such that for every $C_j \in \mathcal{C}$ it holds that C_j is not a subset of S_1 and C_j is not a subset of S_2 ?

First, we describe the reduction from Set Splitting. Given an instance (S, \mathcal{C}) , where we assume without loss of generality that $\bigcup_{C_j \in \mathcal{C}} C_j = S$, construct a nondense set \mathcal{T} of rooted triplets having a leaf set L with $L = \{h, x, y\} \cup \{s_i^j \mid s_i \in S, 1 \leq j \leq m\}$, where h, x, y , and all elements of the form s_i^j are new elements not belonging to S . Initially, let \mathcal{T} consist of the two rooted triplets $(\{x, h\}, y)$ and $(\{y, h\}, x)$. Next, for each $C_j \in \mathcal{C}$, write $C_j = \{s_a, s_b, s_c\}$ with $a < b < c$ and include three rooted triplets $(\{s_a^j, h\}, s_b^j)$, $(\{s_b^j, h\}, s_c^j)$, and $(\{s_c^j, h\}, s_a^j)$ in \mathcal{T} . Finally, for each $s_i \in S$, add m rooted triplets $(\{s_i^1, s_i^2\}, h)$, $(\{s_i^2, s_i^3\}, h), \dots, (\{s_i^{m-1}, s_i^m\}, h)$, $(\{s_i^m, s_i^1\}, h)$ and $2m$ rooted triplets $(\{s_i^1, h\}, x)$, $(\{y, h\}, s_i^1)$, $(\{s_i^2, h\}, x)$, $(\{y, h\}, s_i^2), \dots, (\{s_i^m, h\}, x)$, $(\{y, h\}, s_i^m)$ to \mathcal{T} . (The main idea in the reduction is to encode \mathcal{C} by rooted triplets of the form $(\{s_a^j, h\}, s_b^j)$ and use other rooted triplets to force any galled network N consistent with \mathcal{T} to have a special structure; see Lemma 17. Then, for each $C_j = \{s_a, s_b, s_c\} \in \mathcal{C}$, at most two of s_a^j, s_b^j, s_c^j can descend from the same clipped merge path from the root in N , inducing a set splitting of S .)

LEMMA 16. *If (S, \mathcal{C}) has a set splitting, then there exists a simple phylogenetic network which is consistent with \mathcal{T} .*

Proof. Let (S_1, S_2) be a set splitting of (S, \mathcal{C}) . Define $S_1^* = \{s_i^j \mid s_i \in S_1, 1 \leq j \leq m\}$ and $S_2^* = \{s_i^j \mid s_i \in S_2, 1 \leq j \leq m\}$. Note that $S_1^* \cup S_2^* \cup \{h, x, y\} = L$. Let O_1 be any ordering of $S_1^* \cup \{x\}$ in which x is the first element, and for every pair of elements of the form s_a^j and s_b^j in S_1^* , if there exists a C_j in \mathcal{C} with $C_j = \{s_a, s_b, s_c\}$ and either $b < a < c$, $a < c < b$, or $c < b < a$, then s_a^j precedes s_b^j . (Except for this requirement, the elements may be ordered arbitrarily in O_1 .) Define an ordering O_2 of $S_2^* \cup \{y\}$ analogously, letting y be the last element in O_2 , respectively. Next, build a simple phylogenetic network N having a root node r and a hybrid node whose child is a leaf labeled by h , where (1) $|S_1^*| + 1$ leaves distinctly labeled by $S_1^* \cup \{x\}$ are attached to the left clipped merge path in order according to O_1 , and $|S_2^*| + 1$ leaves distinctly labeled by $S_2^* \cup \{y\}$ are attached to the right clipped merge path in order according to O_2 . See Figure 7 for an example. It is easy to verify that N and \mathcal{T} are consistent. \square

To prove the other direction (i.e., that a galled network consistent with \mathcal{T} yields a set splitting of (S, \mathcal{C})), we need the following lemma.

LEMMA 17. *Suppose N is a galled network with leaf set L which is consistent with \mathcal{T} . Then (1) the root r of N is a split node, (2) one side network attached to a merge path of r contains h but no other leaves, and (3) h is a descendant of the hybrid node for r .*

Proof. (1) Suppose r is not a split node. Then L can be partitioned into two disjoint, nonempty subsets U and V such that every path between a leaf in U and a leaf in V passes through r . It follows that for any rooted triplet $(\{a, b\}, c)$ which is consistent with N , if $a \in U$, then $b \in U$, and if $a \in V$, then $b \in V$. Now consider any element of the form s_i^j in L . If $s_i^j \in U$, then $h, x, y \in U$ because N is consistent with $(\{s_i^j, h\}, x)$, $(\{x, h\}, y)$, and $(\{y, h\}, s_i^j)$. But then also $s_z^k \in U$ for every s_z^k in L by $(\{s_z^k, h\}, x) \in \mathcal{T}$, contradicting that V is nonempty. The case $s_i^j \in V$ is analogous. Hence, r must be a split node.

(2) Let N' be the side network attached to a merge path of r such that $h \in \Lambda(N')$.

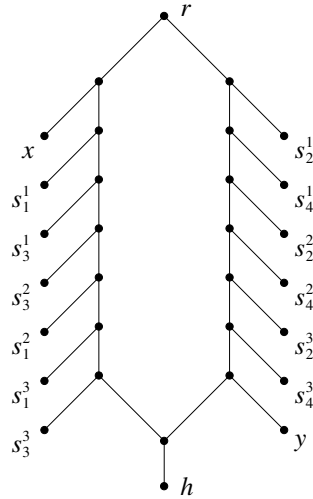


FIG. 7. Let $C_1 = \{s_1, s_2, s_3\}$, $C_2 = \{s_1, s_3, s_4\}$, and $C_3 = \{s_2, s_3, s_4\}$, and suppose $S_1 = \{s_1, s_3\}$ and $S_2 = \{s_2, s_4\}$. The construction described in Lemma 16 yields a simple phylogenetic network N as shown above.

If $x \in \Lambda(N')$, then $\Lambda(N') = L$ because $y \in \Lambda(N')$ by $(\{y, h\}, x) \in \mathcal{T}$ and for every element of the form s_i^j in L , it holds that $s_i^j \in \Lambda(N')$ by $(\{s_i^j, h\}, x) \in \mathcal{T}$. If $y \in \Lambda(N')$, then $\Lambda(N') = L$ because of $x \in \Lambda(N')$ by $(\{x, h\}, y) \in \mathcal{T}$ and the above. If $\Lambda(N')$ contains an element of the form s_i^j , then $\Lambda(N') = L$ because of $y \in \Lambda(N')$ by $(\{y, h\}, s_i^j) \in \mathcal{T}$ and the above. Thus, if $\Lambda(N')$ contains any element in addition to h , then $\Lambda(N') = L$, which is not possible. Therefore, $\Lambda(N') = \{h\}$.

(3) By (1), r is a split node of N . Let $hn(r)$ be the hybrid node for r and let N' be the subnetwork of N rooted at $hn(r)$. Suppose, on the contrary, that h is not contained in N' . Take any $C_j \in \mathcal{C}$ and write $C_j = \{s_a, s_b, s_c\}$ with $a < b < c$. Since $(\{s_a^j, h\}, s_b^j)$, $(\{s_b^j, h\}, s_c^j)$, and $(\{s_c^j, h\}, s_a^j)$ are in \mathcal{T} and $h \notin \Lambda(N')$, exactly one of s_a^j , s_b^j , and s_c^j belongs to $\Lambda(N')$. Assume without loss of generality that $s_a^j \in \Lambda(N')$. Next, neither x nor y can belong to the same side network as h by (2), and since $(\{x, h\}, y)$ and $(\{y, h\}, x)$ are consistent with N , we have either $x \in \Lambda(N')$ and $y \notin \Lambda(N')$ or $x \notin \Lambda(N')$ and $y \in \Lambda(N')$. If $x \in \Lambda(N')$, then $(\{s_a^j, h\}, x)$ is not consistent with N , and if $y \in \Lambda(N')$, then $(\{y, h\}, s_a^j)$ is not consistent with N , which is a contradiction in both cases. Therefore, $h \in \Lambda(N')$. \square

LEMMA 18. *If there exists a galled network which is consistent with \mathcal{T} , then (S, \mathcal{C}) has a set splitting.*

Proof. Let N be a galled network with leaf set L that is consistent with \mathcal{T} . By Lemma 17, the root r of N is a split node. Also by Lemma 17, the subnetwork of N rooted at the child of $hn(r)$, where $hn(r)$ denotes the hybrid node for r , consists of a single leaf which is labeled by h . Let P_1 and P_2 be the two clipped merge paths of $hn(r)$, and define L_1 and L_2 as the set of all leaves except x , y , and h which are descendants of nodes on P_1 and P_2 , respectively. We now show that (L_1, L_2) induces a set splitting (S_1, S_2) of (S, \mathcal{C}) . For every $s_i \in S$, if s_i^1 belongs to L_1 , then all elements in $\{s_i^k \mid 1 \leq k \leq m\}$ belong to L_1 because $(\{s_i^1, s_i^2\}, h)$, $(\{s_i^2, s_i^3\}, h)$, \dots , $(\{s_i^{m-1}, s_i^m\}, h)$, $(\{s_i^m, s_i^1\}, h) \in \mathcal{T}$. Similarly, if $s_i^1 \in L_2$, then $\{s_i^k \mid 1 \leq k \leq m\} \subseteq L_2$. Define $S_1 = \{s_i \mid s_i^1 \in L_1\}$ and $S_2 = \{s_i \mid s_i^1 \in L_2\}$. Clearly, $S_1 \cup S_2 = S$ and $S_1 \cap S_2 = \emptyset$. Moreover, for every $C_j \in \mathcal{C}$, it holds that $C_j \not\subseteq S_1$ and $C_j \not\subseteq S_2$ (to

see this, write $C_j = \{s_a, s_b, s_c\}$ with $a < b < c$ and note that if all of $s_a, s_b,$ and s_c belonged to just one of S_1 and S_2 , then $s_a^j, s_b^j,$ and s_c^j would be descendants of nodes on the same clipped merge path of $hn(r)$, and then all three rooted triplets $(\{s_a^j, h\}, s_b^j), (\{s_b^j, h\}, s_c^j),$ and $(\{s_c^j, h\}, s_a^j)$ could never be consistent with N , which is a contradiction). Thus, (S_1, S_2) is a set splitting of (S, \mathcal{C}) . \square

THEOREM 7. *Given any nondense set \mathcal{T} of rooted triplets, it is NP-hard to determine if there exists a galled network which is consistent with \mathcal{T} . It is also NP-hard to determine if there exists a simple network which is consistent with \mathcal{T} .*

Proof. If (S, \mathcal{C}) has a set splitting, then there exists a simple network which is consistent with \mathcal{T} according to Lemma 16. Next, Lemma 18 shows that if there exists a galled network consistent with \mathcal{T} , then (S, \mathcal{C}) has a set splitting. Since a simple phylogenetic network is always a galled network, and the reduction can be carried out in polynomial time, the theorem follows. \square

5. Approximating the maximum number of consistent rooted triplets.

This section studies the problem of constructing a galled network consistent with the maximum number of rooted triplets in \mathcal{T} for any (not necessarily dense) given \mathcal{T} . Section 5.2 presents a polynomial-time approximation algorithm for this problem which always outputs a galled network consistent with at least a factor of $\frac{5}{12}$ (> 0.4166) of the rooted triplets in \mathcal{T} . On the negative side, section 5.1 shows that there exist inputs for which any galled network can be consistent with at most a factor of 0.4883 of the rooted triplets in \mathcal{T} .

5.1. Inapproximability result. Given any positive integer n , fix \mathcal{T} to contain all possible rooted triplets for a leaf set L of size n , that is, $\mathcal{T} = \{(\{a, b\}, c), (\{a, c\}, b), (\{b, c\}, a) \mid a, b, c \in L\}$. For any phylogenetic network N , let $\#N$ denote the number of rooted triplets from \mathcal{T} that are consistent with N .

LEMMA 19. *Let N be a galled network with $\Lambda(N) = L$. If N contains a nonsplit node u with two children u_1, u_2 such that u is the root node or a child of a hybrid node, then making u into a split node by removing the edges (u, u_1) and (u, u_2) , adding two new nodes v and w , and inserting the edges $(u, v), (u, w), (v, u_1), (w, u_2),$ and (v, w) yields a galled network N' with $\#N' = \#N$.*

LEMMA 20. *Let N be a galled network with $\Lambda(N) = L$. Suppose N contains a merge path P of a hybrid node h, c is the child of $h, N[u]$ is a side network attached to $P, u \neq c,$ and u has two children u_1, u_2 . Then one of the following holds:*

- *If $|\Lambda(N[u])| > |\Lambda(N[c])|,$ then N can be transformed into a galled network N' with $\#N' > \#N$ by letting $N[u]$ and $N[c]$ trade places.*
- *Else, if $|\Lambda(N[u])| \leq |\Lambda(N[c])|,$ then N can be transformed into a galled network N' with $\#N' > \#N$ as follows: First, in case u is a split node in $N,$ delete a hybrid edge descending from u (and contract all edges from vertices with outdegree 1) so that $N[u_1]$ and $N[u_2]$ become disjoint. Second, remove $N[u]$ and instead attach the resulting disjoint $N[u_1]$ and $N[u_2]$ to P .*

Proof. Let s be the split node corresponding to $h,$ and define $L_u = \Lambda(N[u]), L_c = \Lambda(N[c]), L_m = \Lambda(N[s]) \setminus (L_u \cup L_c),$ and $L_{rest} = L \setminus (L_u \cup L_c \cup L_m)$. First consider the case $|\Lambda(N[u])| > |\Lambda(N[c])|.$ For any subset $\{x, y, z\}$ of $L,$ we have the following possibilities:

- When at least one of $\{x, y, z\}$ belongs to $L_{rest},$ it is easy to see that N and N' are consistent with exactly the same rooted triplets labeled by $\{x, y, z\}.$
- When $\{x, y, z\}$ contains at least two leaves from L_u or at least two leaves from $L_c,$ or when $\{x, y, z\}$ contains three leaves from $L_m,$ then N and N' are

again consistent with exactly the same rooted triplets labeled by $\{x, y, z\}$.

- When $\{x, y, z\}$ contains one leaf from L_u , one leaf from L_c , and one leaf from L_m , then each of N and N' is consistent with exactly two rooted triplets labeled by $\{x, y, z\}$.
- When $\{x, y, z\}$ contains one leaf from L_u and two leaves from L_m , then N is consistent with one rooted triplet labeled by $\{x, y, z\}$, whereas N' is consistent with two.
- When $\{x, y, z\}$ contains one leaf from L_c and two leaves from L_m , then N is consistent with two rooted triplets labeled by $\{x, y, z\}$, whereas N' is consistent with one.

Since $|L_u| > |L_c|$, the difference in number of consistent rooted triplets is given by $\#N' - \#N = |L_u| \cdot \binom{|L_m|}{2} - |L_c| \cdot \binom{|L_m|}{2} > 0$.

Next consider the case $|\Lambda(N[u])| \leq |\Lambda(N[c])|$. If u is a split node in N , then delete a hybrid edge e descending from u so that the resulting $\Lambda(N[u_1])$ and $\Lambda(N[u_2])$ are disjoint; assume without loss of generality that e is a descendant of u_2 . In addition to the above, define $L_{u_1} = \Lambda(N[u_1])$ and $L_{u_2} = \Lambda(N[u]) \setminus \Lambda(N[u_1])$. For any subset $\{x, y, z\}$ of L , we have the following possibilities:

- When at least one of $\{x, y, z\}$ belongs to L_{rest} , then N and N' are consistent with exactly the same rooted triplets labeled by $\{x, y, z\}$.
- When $\{x, y, z\}$ does not contain at least one leaf from L_{u_1} and at least one leaf from L_{u_2} , then N and N' are consistent with the same number of rooted triplets labeled by $\{x, y, z\}$.
- When $\{x, y, z\}$ contains one leaf from L_{u_1} , one leaf from L_{u_2} , and one leaf from L_m , then each of N and N' is consistent with one rooted triplet labeled by $\{x, y, z\}$.
- When $\{x, y, z\}$ contains one leaf from L_{u_1} , one leaf from L_{u_2} , and one leaf from L_c , then N is consistent with one rooted triplet labeled by $\{x, y, z\}$, whereas N' is consistent with two.
- When $\{x, y, z\}$ contains two leaves from L_{u_1} and one leaf from L_{u_2} , or one leaf from L_{u_1} and two leaves from L_{u_2} , then N is consistent with one or two rooted triplets labeled by $\{x, y, z\}$, whereas N' is consistent with one.

Since $|L_c| \geq |L_u|$, $|L_u| \geq |L_{u_1}|$, and $|L_u| \geq |L_{u_2}|$, the difference in number of consistent rooted triplets satisfies $\#N' - \#N \geq |L_{u_1}| \cdot |L_{u_2}| \cdot |L_c| - \binom{|L_{u_1}|}{2} \cdot |L_{u_2}| - |L_{u_1}| \cdot \binom{|L_{u_2}|}{2} \geq \frac{1}{2} \cdot |L_{u_1}| \cdot |L_{u_2}| \cdot (2 \cdot |L_u| - |L_{u_1}| - |L_{u_2}| + 2) > 0$. \square

By repeatedly applying Lemmas 19 and 20, the next lemma concludes that for any fixed n , at least one of the galled networks N for a set of n leaves that maximizes $\#N$ must be a *caterpillar network*. A galled network N is called a caterpillar network if (1) the root of N and every nonleaf child of a hybrid node are split nodes and (2) for every merge path P in N , all side networks attached to P except for the one at the hybrid node are leaves.

LEMMA 21. *For any galled network N , there is a caterpillar network N' with $\Lambda(N') = \Lambda(N)$ and $\#N' \geq \#N$.*

Now, we are ready to show the bound on the approximation ratio. Let $S(n)$ be the maximum value of $\#N$ taken over all galled networks N with n leaves.

LEMMA 22. $S(n) = \max_{1 \leq a \leq n} \left\{ \binom{a}{3} + 2 \cdot \binom{a}{2} \cdot (n - a) + a \cdot \binom{n-a}{2} + S(n - a) \right\}$.

Proof. By Lemma 21, there is a caterpillar network N that maximizes $\#N$ among all galled networks with n leaves. The recurrence for $S(n)$ counts the maximum number of rooted triplets in \mathcal{T} consistent with a caterpillar network with n leaves because if such a network contains a set A of a leaves attached to the two merge

paths starting at the root, then it must be consistent with

- $\binom{a}{3}$ rooted triplets labeled by three elements in A ;
- $2 \cdot \binom{a}{2} \cdot (n - a)$ rooted triplets labeled by two elements in A and one element in $L \setminus A$;
- $a \cdot \binom{n-a}{2}$ rooted triplets labeled by one element in A and two elements in $L \setminus A$;
- $S(n - a)$ rooted triplets labeled by three elements in $L \setminus A$. \square

THEOREM 8. *There is no approximation algorithm with approximation ratio larger than 0.4883.*

Proof. Define $T(n) = |\mathcal{T}|$ for any given positive integer n , i.e., $T(n) = 3 \cdot \binom{n}{3}$. Note that the approximation ratio can be at most $\min_{n \in \mathbb{Z}^+} \frac{S(n)}{T(n)}$. By inserting $n = 1000$ into the recurrence in Lemma 22, we obtain $S(1000) = 243383298$. Hence, the approximation ratio must be less than or equal to $\frac{S(1000)}{T(1000)} < 0.4883$. \square

5.2. A polynomial-time $\frac{5}{12}$ -approximation algorithm. Given any set \mathcal{T} of rooted triplets, our approximation algorithm called *Approximate* (shown in Figure 8) infers a galled network which is consistent with at least $\frac{5}{12}$ of the rooted triplets in \mathcal{T} . We first describe the algorithm and then present the analysis.

Initially, *Approximate* partitions the set of leaves L into three subsets A, B, C so that none of them equals L using an algorithm named *LeafPartition* (also listed in Figure 8 and described in detail below). Then, for each $X \in \{A, B, C\}$, it recursively infers a galled network K_X by calling *Approximate*($\mathcal{T}|X$). Next, for each $X \in \{A, B, C\}$, it generates a galled network $Network_X$ such that the root node is a split node whose hybrid node is the parent of K_X , and the other two networks in $\{K_A, K_B, K_C\} \setminus \{K_X\}$ are side networks. Finally, it returns the best network among $Network_A, Network_B,$ and $Network_C$.

We now explain the algorithm *LeafPartition*. It divides L into the three subsets A, B, C in such a way that a special condition $5N_1 + 8N_2 + 12N_3 \geq 5 \cdot |\mathcal{T}|$ holds, where for $i \in \{0, 1, 2, 3\}$, we define $N_i = |Z_i(A, B, C)|$ and where $Z_i(A, B, C)$ is the set defined as follows:

- $Z_0(A, B, C) = \{(\{x, y\}, z) \in \mathcal{T} \mid x \text{ and } z \text{ are in one of the subsets } A, B, C \text{ and } y \text{ is in another}\};$
- $Z_1(A, B, C) = \{(\{x, y\}, z) \in \mathcal{T} \mid x, y, \text{ and } z \text{ are in one of the subsets } A, B, C\};$
- $Z_2(A, B, C) = \{(\{x, y\}, z) \in \mathcal{T} \mid x, y, \text{ and } z \text{ are in three different subsets among } A, B, C\};$
- $Z_3(A, B, C) = \{(\{x, y\}, z) \in \mathcal{T} \mid x \text{ and } y \text{ are in one of the subsets } A, B, C \text{ and } z \text{ is in another}\}.$

Note that $Z_0(A, B, C) \cup Z_1(A, B, C) \cup Z_2(A, B, C) \cup Z_3(A, B, C) = \mathcal{T}$. As shown below, any A, B, C which imply $5N_1 + 8N_2 + 12N_3 \geq 5 \cdot |\mathcal{T}|$ guarantee a good approximation ratio for *Approximate*. Algorithm *LeafPartition* is a greedy algorithm which first divides L into the three subsets arbitrarily and then moves leaves (one at a time) from one subset to another until $score(A, B, C)$ cannot be further improved, where we define $score(A, B, C) = 4N_1 + 7N_2 + 12N_3$. If one of the subsets, say A , equals L after finishing moving the leaves, then it selects a leaf u that maximizes $\frac{p(u)}{c(u)}$, where $p(u) = |\{(\{x, y\}, u) \in \mathcal{T}\}|$ and $c(u) = |\{(\{u, x\}, y) \in \mathcal{T}\}|$, and moves u from A to either B or C . (This step is to ensure that none of the three subsets equals L .) The next lemma shows that this extra move does not reduce the value of $score(A, B, C)$. Since $score$ keeps increasing by at least 1 as long as the while-loop iterates and $score(A, B, C) \leq 12 \cdot |\mathcal{T}|$, step 2.1 is performed at most $12 \cdot |\mathcal{T}|$ times in total; i.e., the algorithm is guaranteed to terminate.

Algorithm *LeafPartition***Input:** A set \mathcal{T} of rooted triplets with a leaf set L .**Output:** A partition of L into three subsets A, B, C such that none of them equals L and such that $5N_1 + 8N_2 + 12N_3 \geq 5 \cdot |\mathcal{T}|$.1 Arbitrarily partition L into three subsets A, B, C .2 **while** moving a leaf m from one subset to another increases $score(A, B, C) = 4N_1 + 7N_2 + 12N_3$ **do**2.1 Move m accordingly.**endwhile**3 **if** one of the subsets A, B, C equals L **then**3.1 Choose a leaf u that maximizes $\frac{p(u)}{c(u)}$ and move u to another subset. Go to Step 2.
endif4 **return** A, B, C .**End** *LeafPartition***Algorithm** *Approximate***Input:** A set \mathcal{T} of rooted triplets with a leaf set L .**Output:** A galled network that is consistent with at least $\frac{5}{12} \cdot |\mathcal{T}|$ of the rooted triplets in \mathcal{T} .1 Partition L into A, B, C using *LeafPartition*.2 For $X \in \{A, B, C\}$, let $K_X = \text{Approximate}(\mathcal{T}|X)$.3 For $X \in \{A, B, C\}$, generate a galled network $Network_X$ in which the root node is a split node whose hybrid node h is the parent of K_X , and the other two networks in $\{K_A, K_B, K_C\} \setminus \{K_X\}$ are side networks attached to the merge paths of h .4 **return** the $Network_X$ among $X \in \{A, B, C\}$ that is consistent with the most rooted triplets in \mathcal{T} .**End** *Approximate*FIG. 8. An approximation algorithm for computing a galled network consistent with as many rooted triplets in \mathcal{T} as possible.

LEMMA 23. *Algorithm LeafPartition partitions L into three subsets A, B, C so that $score(A, B, C)$ cannot be further improved by moving a single element from one subset to another.*

Proof. If none of A, B, C equals L after step 2 in Algorithm *LeafPartition* is done, the lemma follows. Hence, assume that one of the subsets, say A , equals L after step 2. We only need to show that step 3.1 does not decrease $score(A, B, C)$. When u is moved from A , all rooted triplets in \mathcal{T} of the form $(\{x, y\}, u)$ are moved from $Z_1(A, B, C)$ to $Z_3(A, B, C)$ and all rooted triplets in \mathcal{T} of the form $(\{u, x\}, y)$ are moved from $Z_1(A, B, C)$ to $Z_0(A, B, C)$. The difference in $score$ is equal to $score(A \setminus \{u\}, \{u\}, \emptyset) - score(A, \emptyset, \emptyset) = p(u) \cdot (12 - 4) - c(u) \cdot 4 \geq 0$, where the last inequality follows since $p(u) \geq \frac{1}{2} \cdot c(u)$ by the choice of u . Thus, step 3.1 will not decrease $score(A, B, C)$. \square

The next two lemmas are needed to analyze the approximation ratio of *Approximate*.

LEMMA 24. *When Algorithm LeafPartition terminates, we have $5N_1 + 8N_2 + 12N_3 \geq 5 \cdot |\mathcal{T}|$.*

Proof. Write $score(A, B, C) = x \cdot N_1 + y \cdot N_2 + z \cdot N_3$. For any $U, V, W \in \{A, B, C\}$, denote by $(\{U, V\}, W)$ the set of all rooted triplets in \mathcal{T} of the form $(\{u, v\}, w)$ where $u \in U$, $v \in V$, and $w \in W$. Similarly, for any $m \in L$ and $U, V \in \{A, B, C\}$, let

$(\{m, U\}, V)$ and $(\{U, V\}, m)$ denote the set of all rooted triplets in \mathcal{T} of the form $(\{m, u\}, v)$ and $(\{u, v\}, m)$, respectively, where $u \in U$ and $v \in V$. For each of the six possible ways of moving a leaf m from one of the subsets A, B, C to another, we can derive a formula to express how *score* is affected, as described next.

First, suppose m is moved from A to B . Then, for every element t in $(\{m, A\}, A)$, since t will be moved from $Z_1(A, B, C)$ to $Z_0(A, B, C)$, the corresponding change in *score* is $-x$. In the same way, we can calculate the change in *score* for each element in $(\{m, U\}, V)$ and $(\{U, V\}, m)$ for every $U, V \in \{A, B, C\}$ when m is moved from A to B . After *LeafPartition* is done, moving m will not increase the value of *score*, so $score(A \setminus \{m\}, B \cup \{m\}, C) - score(A, B, C) \leq 0$. Thus, we have $-x|(\{m, A\}, A)| - z|(\{m, A\}, B)| + (y - z)|(\{m, A\}, C)| + z|(\{m, B\}, A)| + x|(\{m, B\}, B)| + (z - y)|(\{m, B\}, C)| + y|(\{m, C\}, A)| - y|(\{m, C\}, B)| + 0 \cdot |(\{m, C\}, C)| + (z - x)|(\{A, A\}, m)| + 0 \cdot |(\{A, B\}, m)| + y|(\{A, C\}, m)| + (x - z)|(\{B, B\}, m)| - y|(\{B, C\}, m)| + 0 \cdot |(\{C, C\}, m)| \leq 0$.

Next, by summing over all $m \in A$, we obtain the following inequality I_{AB} :

$$I_{AB}: -2x|(\{A, A\}, A)| - 2z|(\{A, A\}, B)| + 2(y - z)|(\{A, A\}, C)| + z|(\{A, B\}, A)| + x|(\{A, B\}, B)| + (z - y)|(\{A, B\}, C)| + y|(\{A, C\}, A)| - y|(\{A, C\}, B)| + (z - x)|(\{A, A\}, A)| + y|(\{A, C\}, A)| + (x - z)|(\{B, B\}, A)| - y|(\{B, C\}, A)| \leq 0.$$

In the summation, each element of the form $(\{a_1, a_2\}, x)$ where $a_1, a_2 \in A$ is counted twice; therefore, the coefficient of each $|(\{A, A\}, x)|$ is multiplied by 2.

We derive five inequalities $I_{AC}, I_{BA}, I_{BC}, I_{CA}$, and I_{CB} analogously. Finally, we add $I_{AB}, I_{AC}, I_{BA}, I_{BC}, I_{CA}$, and I_{CB} together, and use $N_0 = |(\{A, B\}, A)| + |(\{A, B\}, B)| + |(\{A, C\}, A)| + |(\{A, C\}, C)| + |(\{B, C\}, B)| + |(\{B, C\}, C)|$, $N_1 = |(\{A, A\}, A)| + |(\{B, B\}, B)| + |(\{C, C\}, C)|$, $N_2 = |(\{A, B\}, C)| + |(\{A, C\}, B)| + |(\{B, C\}, A)|$, and $N_3 = |(\{A, A\}, B)| + |(\{A, A\}, C)| + |(\{B, B\}, A)| + |(\{B, B\}, C)| + |(\{C, C\}, A)| + |(\{C, C\}, B)|$ to obtain $(z + 2y + x) \cdot N_0 + (2z - 6x) \cdot N_1 + (2z - 6y) \cdot N_2 + (2y + x - 5z) \cdot N_3 \leq 0$. By substituting $N_0 = |\mathcal{T}| - N_1 - N_2 - N_3$ and replacing $x = 4, y = 7, z = 12$, we get $5N_1 + 8N_2 + 12N_3 \geq 5 \cdot |\mathcal{T}|$. \square

Let $m(\mathcal{T})$ be the number of rooted triplets in \mathcal{T} consistent with the network returned by *Approximate*(\mathcal{T}).

LEMMA 25. *If $m(\mathcal{T}|Z) \geq q \cdot |\mathcal{T}|Z|$ for every $Z \in \{A, B, C\}$, then $m(\mathcal{T}) \geq q \cdot N_1 + \frac{2}{3} \cdot N_2 + N_3$.*

Proof. Every rooted triplet in $Z_2(A, B, C)$ is consistent with two of $Network_A, Network_B$, and $Network_C$, and every rooted triplet in $Z_3(A, B, C)$ is consistent with all of these three networks. Thus, $Network_X$ returned by *Approximate* must be consistent with at least $\frac{2}{3} \cdot N_2 + N_3$ of the rooted triplets in $Z_2(A, B, C) \cup Z_3(A, B, C)$. Also, each of $Network_A, Network_B$, and $Network_C$ is consistent with $m(\mathcal{T}|A) + m(\mathcal{T}|B) + m(\mathcal{T}|C) \geq q \cdot (|\mathcal{T}|A| + |\mathcal{T}|B| + |\mathcal{T}|C|) = q \cdot N_1$ of the rooted triplets in $Z_1(A, B, C)$. Thus, in total, $Network_X$ is consistent with at least $q \cdot N_1 + \frac{2}{3} \cdot N_2 + N_3$ rooted triplets in \mathcal{T} . \square

THEOREM 9. $m(\mathcal{T}) \geq \frac{5}{12} \cdot |\mathcal{T}|$.

Proof. By induction on $|L|$. Base case ($|L| = 3$): Steps 3 and 4 of Algorithm *Approximate* construct a network consistent with at least $2/3$ of the rooted triplets in \mathcal{T} ; i.e., $m(\mathcal{T}) \geq \frac{5}{12} \cdot |\mathcal{T}|$. Inductive case ($|L| > 3$) Step 2 of *Approximate* recursively constructs three networks K_A, K_B, K_C for $\mathcal{T}|A, \mathcal{T}|B$, and $\mathcal{T}|C$, respectively. By the induction assumption, $m(\mathcal{T}|X) \geq \frac{5}{12} \cdot |\mathcal{T}|X|$ for each $X \in \{A, B, C\}$. By Lemmas 24 and 25, $m(\mathcal{T}) \geq \frac{5}{12} \cdot N_1 + \frac{2}{3} \cdot N_2 + N_3 \geq \frac{5}{12} \cdot |\mathcal{T}|$. \square

Finally, the algorithm's running time is given by the following theorem.

THEOREM 10. *The time complexity of Algorithm Approximate is $O(n \cdot |\mathcal{T}|^3)$.*

Proof. Denote by $t(\mathcal{T})$ and $f(\mathcal{T})$ the running times of *Approximate*(\mathcal{T}) and *LeafPartition*(\mathcal{T}), respectively. We have $t(\mathcal{T}) = f(\mathcal{T}) + t(\mathcal{T}|A) + t(\mathcal{T}|B) + t(\mathcal{T}|C)$.

In *LeafPartition*, step 2 is performed at most $12 \cdot |\mathcal{T}|$ times in total. Every time, the algorithm needs to compute $O(n)$ values of *score*, and each *score* can be computed in $O(|\mathcal{T}|)$ time. Steps 1 and 3 can easily be implemented in $O(|\mathcal{T}|)$ time. Therefore, $f(\mathcal{T}) = O(n \cdot |\mathcal{T}|^2)$.

Furthermore, $|\mathcal{T}|A + |\mathcal{T}|B + |\mathcal{T}|C < |\mathcal{T}|$. Solving the recurrence for $t(\mathcal{T})$ gives us $t(\mathcal{T}) = O(n \cdot |\mathcal{T}|^3)$. \square

Acknowledgment. We thank Ho-Leung Chan for his comments on this paper.

REFERENCES

- [1] A. V. AHO, Y. SAGIV, T. G. SZYMANSKI, AND J. D. ULLMAN, *Inferring a tree from lowest common ancestors with an application to the optimization of relational expressions*, SIAM J. Comput., 10 (1981), pp. 405–421.
- [2] D. BRYANT, *Building Trees, Hunting for Trees, and Comparing Trees: Theory and Methods in Phylogenetic Analysis*, Ph.D. thesis, University of Canterbury, Christchurch, New Zealand, 1997.
- [3] B. CHOR, M. HENDY, AND D. PENNY, *Analytic solutions for three-taxon ML_{MC} trees with variable rates across sites*, in Proceedings of the 1st Annual Workshop on Algorithms in Bioinformatics (WABI 2001), Lecture Notes in Comput. Sci. 2149, Springer-Verlag, Berlin, 2001, pp. 204–213.
- [4] C. CHOY, J. JANSSON, K. SADAKANE, AND W.-K. SUNG, *Computing the maximum agreement of phylogenetic networks*, Theoret. Comput. Sci., 335 (2005), pp. 93–107.
- [5] M. GAREY AND D. JOHNSON, *Computers and Intractability—A Guide to the Theory of NP-Completeness*, W. H. Freeman, New York, 1979.
- [6] L. GAŚSIENIEC, J. JANSSON, A. LINGAS, AND A. ÖSTLIN, *Inferring ordered trees from local constraints*, in Proceedings of the 4th Annual Australasian Theory Symposium of Computing (CATS'98), Aust. Comput. Sci. Commun. 20, Springer-Verlag, Singapore, 1998, pp. 67–76.
- [7] L. GAŚSIENIEC, J. JANSSON, A. LINGAS, AND A. ÖSTLIN, *On the complexity of constructing evolutionary trees*, J. Comb. Optim., 3 (1999), pp. 183–197.
- [8] D. GUSFIELD, S. EDDHU, AND C. LANGLEY, *Optimal, efficient reconstruction of phylogenetic networks with constrained recombination*, J. Bioinform. Comput. Biol., 2 (2004), pp. 173–213.
- [9] J. HEIN, *Reconstructing evolution of sequences subject to recombination using parsimony*, Math. Biosci., 98 (1990), pp. 185–200.
- [10] M. R. HENZINGER, V. KING, AND T. WARNOW, *Constructing a tree from homeomorphic subtrees, with applications to computational evolutionary biology*, Algorithmica, 24 (1999), pp. 1–13.
- [11] J. HOLM, K. DE LICHTENBERG, AND M. THORUP, *Poly-logarithmic deterministic fully-dynamic algorithms for connectivity, minimum spanning tree, 2-edge, and biconnectivity*, J. ACM, 48 (2001), pp. 723–760.
- [12] D. H. HUSON, T. DEZULIAN, T. KLÖPPER, AND M. STEEL, *Phylogenetic super-networks from partial trees*, in Proceedings of the 4th Annual Workshop on Algorithms in Bioinformatics (WABI 2004), Lecture Notes in Comput. Sci. 3240, Springer-Verlag, Berlin, 2004, pp. 388–399.
- [13] J. JANSSON, *On the complexity of inferring rooted evolutionary trees*, in Proceedings of the Brazilian Symposium on Graphs, Algorithms, and Combinatorics (GRACO 2001), Electron. Notes Discrete Math. 7, Elsevier, 2001, pp. 121–125.
- [14] J. JANSSON, J. H.-K. NG, K. SADAKANE, AND W.-K. SUNG, *Rooted maximum agreement supertrees*, Algorithmica, 43 (2005), pp. 293–307.
- [15] J. JANSSON AND W.-K. SUNG, *Inferring a level-1 phylogenetic network from a dense set of rooted triplets*, Theoret. Comput. Sci., to appear.
- [16] T. JIANG, P. KEARNEY, AND M. LI, *A polynomial time approximation scheme for inferring evolutionary trees from quartet topologies and its application*, SIAM J. Comput., 30 (2001), pp. 1942–1961.
- [17] S. KANNAN, E. LAWLER, AND T. WARNOW, *Determining the evolutionary tree using experiments*, J. Algorithms, 21 (1996), pp. 26–50.

- [18] P. KEARNEY, *Phylogenetics and the quartet method*, in Current Topics in Computational Molecular Biology, T. Jiang, Y. Xu, and M. Q. Zhang, eds., The MIT Press, Cambridge, MA, 2002, pp. 111–133.
- [19] L. NAKHLEH, T. WARNOW, C. R. LINDER, AND K. ST. JOHN, *Reconstructing reticulate evolution in species—Theory and practice*, J. Comput. Biol., 12 (2005), pp. 796–811.
- [20] M. P. NG, M. STEEL, AND N. C. WORMALD, *The difficulty of constructing a leaf-labelled tree including or avoiding given subtrees*, Discrete Appl. Math., 98 (2000), pp. 227–235.
- [21] M. P. NG AND N. C. WORMALD, *Reconstruction of rooted trees from subtrees*, Discrete Appl. Math., 69 (1996), pp. 19–31.
- [22] D. POSADA AND K. A. CRANDALL, *Intraspecific gene genealogies: Trees grafting into networks*, Trends Ecol. Evol., 16 (2001), pp. 37–45.
- [23] M. STEEL, *The complexity of reconstructing trees from qualitative characters and subtrees*, J. Classification, 9 (1992), pp. 91–116.
- [24] L. WANG, K. ZHANG, AND L. ZHANG, *Perfect phylogenetic networks with recombination*, J. Comput. Biol., 8 (2001), pp. 69–78.
- [25] B. Y. WU, *Constructing the maximum consensus tree from rooted triples*, J. Comb. Optim., 8 (2004), pp. 29–39.

CERTIFYING POLYNOMIAL TIME AND LINEAR/POLYNOMIAL SPACE FOR IMPERATIVE PROGRAMS*

KARL-HEINZ NIGGL[†] AND HENNING WUNDERLICH[†]

Abstract. In earlier work of Kristiansen and Niggl the polynomial-time computable functions were characterized by stack programs of μ -measure 0, and the linear-space computable functions by loop programs of μ -measure 0. Until recently, an open problem was how to extend these characterizations to programs with user-friendly basic instructions, such as assignment statements, and with mixed data structures.

It is shown how to strengthen the above characterizations to imperative programs built from arbitrary basic instructions by sequencing and by if-then-else and for-do statements. These programs operate on variables, each of which may represent any data structure such as stacks, registers, trees, or graphs.

The paper presents a new method of certifying “polynomial size boundedness” of such imperative programs under the natural assumption that the basic instructions used are polynomially size bounded, too. The certificate for a program P with variables among X_1, \dots, X_n will be an $(n+1) \times (n+1)$ matrix $M(P)$ over the finite set $\{0, 1, \infty\}$.

It is shown that certified string programs (i.e., stack programs, but with any polynomial-time computable basic instructions) exactly compute the functions in $FPTIME$. Accordingly, certified general loop programs (using any linear-space computable basic instructions) exactly compute the functions in $FLINSPACE$.

Furthermore, it is shown that certified power string programs (i.e., string programs, but built from polynomial-space computable basic instructions and extended by power loop statements) exactly compute the polynomial-space computable functions in $FPSPACE$.

In addition, examples of certified “natural” (implementations of) algorithms, such as insertion-sort or binary addition and multiplication, are given.

Key words. polynomial time, linear space, polynomial space, static program analysis, property testing, implicit computational complexity, imperative programming languages

AMS subject classifications. 68Q15, 68Q25, 68Q60, 03D15

DOI. 10.1137/S0097539704445597

1. Introduction. In recent research [16], [29] the polynomial-time computable functions are characterized by stack programs of μ -measure 0, and loop programs of μ -measure 0 exactly compute the linear-space computable functions. Loop programs are a slight modification of the LOOP programs of Meyer and Ritchie [25], and stack programs are essentially loop programs that operate with stacks instead of registers, supporting a suitable loop concept over stacks. The measure μ is a conceptually simple, purely syntactical method of analyzing the impact of nesting loops on the running time, and it associates to each program a natural number such that programs of μ -measure $n \geq 1$ exactly compute those functions computed by a Turing machine whose running time is in Grzegorzcyk class \mathcal{E}_{n+2} [12].

From a programming perspective, these findings might not be practically appealing, for unlike modern programming languages, those programs support neither user-friendly basic instructions nor mixed data structures.

Although it is no problem to extend the measure μ to programs with any nonsize-increasing basic instructions (cf. [16]), until recently, an open problem was how to

*Received by the editors August 9, 2004; accepted for publication (in revised form) August 25, 2005; published electronically March 3, 2006.

<http://www.siam.org/journals/sicomp/35-5/44559.html>

[†]Institut für Theoretische Informatik, Technische Universität Ilmenau, Helmholtzplatz 1, 98684 Ilmenau, Germany (niggl@tu-ilmenau.de, henning.wunderlich@tu-ilmenau.de).

extend the above characterizations to programs with arbitrary size-increasing basic instructions or vital instructions, such as assignment statements, without losing too many programs that could be certified as, e.g., running in polynomial time.

In this paper, we show how to strengthen the above characterizations to *imperative programs* built from arbitrary basic instructions by sequencing and by if-then-else and for-do statements. Each of those programs operates on finitely many variables X_1, \dots, X_n , each of which may represent any data structure such as stacks, registers, trees, or graphs, and is equipped implicitly with a notion of *size* of an object stored in X_i , denoted by $|X_i|$. For example, if X_i serves as a register, then $|X_i|$ might be the unary or the binary length of the number stored in X_i , and if X_i serves as a stack, $|X_i|$ is, as usual, the length of the word stored in X_i .

The paper presents a new method of certifying “polynomial size boundedness” for such programs under the natural assumption that all basic instructions are *polynomially size bounded*, too. Expressed in Hoare-like sentences $\{A\} P \{B\}$, for programs P in variables X_1, \dots, X_n , that means there exist polynomials p_1, \dots, p_n satisfying

$$\{s_1 = |X_1|, \dots, s_n = |X_n|\} P \{|X_i| \leq p_i(s_1, \dots, s_n)\} \text{ for } i = 1, \dots, n.$$

Thus, unlike the measure μ , the new method abstracts from the concrete form of basic instructions and focuses on their impact on the polynomial size bounds on the variables involved. As we shall see, polynomial size bounds provide all information on the “control” of one variable over another in a much more subtle way than the measure μ does. Central to the method is that we store and process only a finite amount of information on the class of possible polynomial size bounds for programs. For each polynomial size bound p on X_i with respect to a program P , say $p(\vec{X}) = c_0 + \dots + c_j \cdot X_1^{j_1} \cdot \dots \cdot X_n^{j_n} + \dots$, that information is just an $(n + 1)$ -tuple $\langle p \rangle$ over $\{0, 1, \infty\}$, where for $j = 1, \dots, n$,

$$\langle p \rangle[j] = \begin{cases} 0 & \text{if } p \text{ is a polynomial in } \vec{X} \setminus X_j, \\ 1 & \text{if } p = X_j + q \text{ for some polynomial } q \text{ in } \vec{X} \setminus X_j, \\ \infty & \text{else,} \end{cases}$$

$$\langle p \rangle[n + 1] = \begin{cases} c_0 & \text{if } c_0 \leq 1, \\ \infty & \text{else.} \end{cases}$$

Thus, the certificate for a program P in variables X_1, \dots, X_n will be an $(n + 1) \times (n + 1)$ matrix $M(P)$ over $\{0, 1, \infty\}$, where for technical reasons the last row $M(P)[n + 1]$ is always the $(n + 1)$ -tuple $(0, \dots, 0, 1)$.

Altogether that results in a *matrix calculus for program certificates*. In particular, that calculus provides criteria on the certificate for the body of a loop which guarantee the existence of a certificate for the loop statement itself. We investigate two forms of loop statements, $\text{loopI } X_h [Q]$ and $\text{loopII } X_h [Q]$, the intuition being that for *loopI statements* the body is executed $|X_h|$ times, while the body of *loopII statements* is executed $2^{|X_h|} - 1$ times.

Strengthening the results for μ -measure 0 programs, the following theorems are obtained.

THEOREM A. *Certified string programs (stack programs built from any polynomial-time computable basic instructions) exactly compute the functions in FPTIME.*

THEOREM B. *Certified general loop programs (loop programs built from any linear-space computable basic instructions) precisely compute the functions in FLINSPACE.*

Extending string programs by power loop statements and admitting any polynomial-space computable basic instructions, we obtain the following result.

THEOREM C. *Certified power string programs exactly compute the polynomial-space computable functions FPSPACE.*

Thus, the novelty of the present certification method is that for a significantly large class of imperative programs very close to those in programming practice, with no restrictions on the basic instructions or the data structures involved (cf. section 8 for examples), programs can be certified so as to run in polynomial time or in linear/polynomial space.

The present paper continues research in implicit computational complexity as initiated by Simmons [33], Bellantoni and Cook [4], and Leivant [18], [19], [20], which has led to resource-free, purely functional characterizations of many complexity classes, such as FPTIME [4], [21], [23], [5], [28], FLINSPACE [2], [20], [5], [28], NP and the polynomial-time hierarchy [3], the Kalmár-elementary functions [30] and FPSPACE [15], [31], the exponential-time functions of linear growth [9], and the Grzegorzcyk hierarchy at and above the linear-space level [5], [28], [16], [17], among many others. As well, implicit characterizations through higher-type recursion have been given for the Kalmár-elementary functions [22], [15], [1], for polynomial space [24], and for FPTIME [6], [13].

There are several groups which work on program verification and property testing, e.g., the MRG group [27], the CRISS group [11], or the “Nancy group” (cf. [8], [7], and [26]). There might be some connection between the present work and those interesting approaches, but due to the different frameworks an exact comparison has not been made.

The paper is organized as follows. In section 2, all basic notions involved in the design of certificates for the class of imperative programs under consideration are introduced. Section 3 is concerned with constructing certificates for composed imperative programs, resulting in the certification method. In section 4, stack and loop programs and the measure μ are reviewed, and it is shown that all programs of μ -measure 0 are certified, too. Section 5 introduces string programs and establishes Theorem A. In section 6, general loop programs are introduced, and a proof of Theorem B is given. Section 7 presents power string programs and the proof of Theorem C. In section 8 “natural” implementations of INSERTION-SORT, BINARY ADDITION, MULTIPLICATION, and EXPONENTIATION are given and certified, except for the latter, where the method correctly fails.

2. Preliminaries. Each program we consider uses finitely many variables X_1, \dots, X_n , each of which may represent any data structure such as stacks, registers, trees, or graphs, and is equipped implicitly with a notion of *size* of an object stored in X_i , denoted by $|X_i|$. For example, if X_i serves as a stack (over a given alphabet), $|X_i|$ is the length of the word stored in X_i .

All we require from the primitives that those programs are built from is that they are polynomially size bounded.

DEFINITION 2.1. *A program P with variables among X_1, \dots, X_n is polynomially size bounded (psb) iff there exist polynomials $p_1(\vec{X}), \dots, p_n(\vec{X})$ such that (expressed in Hoare notation)*

$$\{s_1 = |X_1|, \dots, s_n = |X_n|\} P \{|X_i| \leq p_i(s_1, \dots, s_n)\} \text{ for } i = 1, \dots, n.$$

Any such list of polynomials is called a polynomial bound on P.

The imperative programs under consideration are built from arbitrary psb *basic instructions* $\text{imp}(\vec{X})$ by sequencing $P_1; P_2$, conditionals $\text{if (cond) then } P_1 \text{ else } P_2$, and two forms of *loops*, $\text{loopI } X_h [Q]$ and $\text{loopII } X_h [Q]$.

We do not specify the particular form of the condition (cond) in a conditional. All we require is that any instance of (cond) be evaluated in polynomial time (in the size of the variables involved). Along these lines, we do not specify the particular form of the two loop statements, but we do specify that the body of loopI is executed $|X_h|$ times, while the body of loopII is executed $2^{|X_h|} - 1$ times. Furthermore, we require that during the execution of a loop statement the contents of its *control variable* X_h remain unchanged.

Throughout this paper, *polynomials* $p(\vec{X})$ in variables X_1, \dots, X_n are expressions

$$p(\vec{X}) = c_0 + \dots + c_i \cdot X_1^{i_1} \dots X_n^{i_n} + \dots$$

with coefficients $c_i \in \mathbb{N}$, and we write $\kappa(p)$ for the *constant* coefficient c_0 . For convenience, we use the same variable names for both polynomials and programs but, no doubt, for polynomials those variables range over natural numbers only.

Given another polynomial $q(\vec{X}) = d_0 + \dots + d_i \cdot X_1^{i_1} \dots X_n^{i_n} + \dots$, the coefficientwise *maximum* of p and q , denoted by $p \sqcup q$, is defined by

$$p \sqcup q := \max(c_0, d_0) + \dots + \max(c_i, d_i) \cdot X_1^{i_1} \dots X_n^{i_n} + \dots$$

Furthermore, we write $X_i \in p$ ($X_i \notin p$) if X_i occurs in p (if p is a polynomial in $\vec{X} \setminus X_i$), and we say that X_i *occurs simple* in p , denoted by $X_i \in s(p)$, if p can be written as

$$p(X_1, \dots, X_n) = X_i + q(X_1, \dots, X_{i-1}, X_{i+1}, \dots, X_n).$$

As pointed out in the introduction, all that we store and process about polynomials $p(X_1, \dots, X_n)$ is whether $X_i \notin p$ (represented by 0) or $X_i \in s(p)$ (represented by 1) or otherwise (represented by ∞), resulting into the *forgetting set* $A := \{0, 1, \infty\}$. This set is ordered by $0 < 1 < \infty$, and the binary operations $+$, \bullet , and \sqcup on A are defined as follows:

$+$	0	1	∞	\bullet	0	1	∞	\sqcup	0	1	∞
0	0	1	∞	0	0	0	0	0	0	1	∞
1	1	∞	∞	1	0	1	∞	1	1	1	∞
∞	∞	∞	∞	∞	0	∞	∞	∞	∞	∞	∞

One can easily verify that these operations are commutative, associative, and distributive (for $+$, \bullet only), and $0, 1, 0$ are the neutral elements of $+$, \bullet , \sqcup , respectively.

DEFINITION 2.2. For $n \geq 1$, let $\mathcal{M}_n[A]$ denote the set of all $n \times n$ matrices M over A , and let $\tilde{\mathcal{M}}_n[A]$ denote the set of all $M \in \mathcal{M}_{n+1}[A]$ with last row $M[n+1] = 0^n 1$.

As usual, one defines a *matrix multiplication* \otimes on $\mathcal{M}_n[A]$, which is associative and has the *identity matrix* 1_n as the neutral element. Let $+$, \bullet , \sqcup denote the componentwise extension of these operations to A^m and $\mathcal{M}_n[A]$, and $<$ the componentwise extension of $<$ to A^m .

3. Constructing certificates for imperative programs. In this section we will define and construct certificates for essentially composed programs.

DEFINITION 3.1. For any polynomial $p(X_1, \dots, X_n)$, let $\langle p \rangle \in A^{n+1}$ be defined by

$$\langle p \rangle[i] := \begin{cases} 0 & \text{if } X_i \notin p, \\ 1 & \text{if } X_i \in s(p), \\ \infty & \text{else,} \end{cases}$$

$$\langle p \rangle[n+1] := \begin{cases} \kappa(p) & \text{if } \kappa(p) \leq 1, \\ \infty & \text{else,} \end{cases}$$

where $i \in \{1, \dots, n\}$. $\langle p \rangle$ is called the representation of p .

DEFINITION 3.2. For $n \geq 0$ and $\vec{a} \in A^{n+1}$, the set of polynomials of bound \vec{a} is

$$\text{POLY}(\vec{a}) := \{p(\mathbf{X}_1, \dots, \mathbf{X}_n) \mid \langle p \rangle \leq \vec{a}\}.$$

To verify that a polynomial belongs to $\text{POLY}(\vec{a})$, one can use the following fact.

COROLLARY 3.3 (POLY explicit). Let $p(\vec{\mathbf{X}})$ be any polynomial, and let $\vec{a} \in A^{n+1}$. Then

$$\begin{aligned} p \in \text{POLY}(\vec{a}) \iff & \forall i \in \{1, \dots, n\}: (a_i = 0 \Rightarrow \mathbf{X}_i \notin p) \wedge \\ & (a_i = 1 \Rightarrow \mathbf{X}_i \in s(p) \vee \mathbf{X}_i \notin p) \\ & \wedge (a_{n+1} = 0 \Rightarrow \mathbf{K}(p) = 0) \\ & \wedge (a_{n+1} = 1 \Rightarrow \mathbf{K}(p) \leq 1). \end{aligned}$$

DEFINITION 3.4 (certificate). Let \mathbf{P} be any program in variables $\mathbf{X}_1, \dots, \mathbf{X}_n$. A certificate for \mathbf{P} is any matrix $Z \in \tilde{\mathcal{M}}_n[A]$ such that there exists a polynomial bound $p_1 \in \text{POLY}(Z[1]), \dots, p_n \in \text{POLY}(Z[n])$ on \mathbf{P} (cf. Definition 2.1).

NOTE 3.5. If \vec{p} is a polynomial bound on the program \mathbf{P} , the matrix $\langle \mathbf{P}, \vec{p} \rangle$ is a certificate for \mathbf{P} , where

$$\langle \mathbf{P}, \vec{p} \rangle := \begin{pmatrix} \langle p_1 \rangle \\ \vdots \\ \langle p_n \rangle \\ 0^n 1 \end{pmatrix}.$$

This is exactly how certificates for basic instructions $\text{imp}(\vec{\mathbf{X}})$ with polynomial bound \vec{p} are constructed (cf. sections 5–8 for examples).

In the following, we tacitly assume that all programs \mathbf{P} have variables among $\mathbf{X}_1, \dots, \mathbf{X}_n$.

DEFINITION 3.6. For $u \in \{0, 1\}^{n+1}$, let q_u be the polynomial induced by u , i.e.,

$$q_u := u[n+1] + \sum_{i=1}^n u[i] \cdot \mathbf{X}_i.$$

LEMMA 3.7 (structure). Let u, v be in A^{n+1} .

(a) If $u \leq v$, then $\text{POLY}(u) \subseteq \text{POLY}(v)$.

(b) If $\infty \notin u$, then $p \leq q_u$ for all $p \in \text{POLY}(u)$.

Proof. (a) is obvious from Def. 3.2; (b) follows from $\langle q_u \rangle = u$ for $u \in \{0, 1\}^{n+1}$. \square

LEMMA 3.8 (conditional). Let v_1, v_2 be in A^{n+1} .

(a) If $p_1 \in \text{POLY}(v_1)$ and $p_2 \in \text{POLY}(v_2)$, then $p_1 \sqcup p_2 \in \text{POLY}(v_1 \sqcup v_2)$.

(b) If Z_1, Z_2 are certificates for $\mathbf{P}_1, \mathbf{P}_2$, respectively, then

- $Z_1 \sqcup Z_2$ is a certificate for the conditional **if** (cond) **then** \mathbf{P}_1 **else** \mathbf{P}_2 ;
- $Z_1 \sqcup 1_{n+1}$ is a certificate for the conditional **if** (cond) **then** \mathbf{P}_1 .

Proof. (b) follows from (a), and (a) follows from $\langle p_1 \rangle \leq v_1 \wedge \langle p_2 \rangle \leq v_2 \Rightarrow \langle p_1 \sqcup p_2 \rangle \leq v_1 \sqcup v_2$. \square

Note that $\sqcup 1_{n+1}$ takes into account the case where \mathbf{P}_1 is not executed.

LEMMA 3.9 (sequence). Let u, v_1, \dots, v_n be in A^{n+1} .

- (a) Let p, q_1, \dots, q_n be polynomials in $\text{POLY}(u), \text{POLY}(v_1), \dots, \text{POLY}(v_n)$, respectively. Then $r \in \text{POLY}(w)$, where $r(\vec{X}) := p(q_1(\vec{X}), \dots, q_n(\vec{X}))$ and

$$w := u \otimes \begin{pmatrix} v_1 \\ \vdots \\ v_n \\ 0^n 1 \end{pmatrix}.$$

- (b) If Z_1, Z_2 are certificates for P_1, P_2 , respectively, then $Z_2 \otimes Z_1$ is a certificate for $P_1; P_2$.

Proof. Part (b) follows from (a). As for the proof of (a), we use Corollary 3.3 and distinguish several cases, where t, t_0, j range over numbers in $\{1, \dots, n\}$.

Case $w[n+1] = 0$. In other words, $u[n+1] \bullet 1 + \sum_{t=1}^n u[t] \bullet v_t[n+1] = 0$; hence $u[n+1] = 0$, and $u[t] = 0$ or $v_t[n+1] = 0$ for all t . By the hypothesis, that implies $\kappa(p) = 0$, and $(X_t \notin p \text{ or } \kappa(q_t) = 0)$ for all t . Thus, $\kappa(r) = 0$ by definition of r .

Case $w[n+1] = 1 = u[n+1] \bullet 1 + \sum_{t=1}^n u[t] \bullet v_t[n+1]$. We distinguish two subcases.

Subcase $u[n+1] = 1$ and $(u[t] = 0 \text{ or } v_t[n+1] = 0)$ for all t . By the hypothesis, that implies $\kappa(p) \leq 1$, and $(X_t \notin p \text{ or } \kappa(q_t) = 0)$ for all t . Hence $\kappa(r) \leq 1$ by definition of r .

Subcase $u[n+1] = 0$, and $u[t_0] = 1 = v_{t_0}[n+1]$ for some t_0 , and $(u[t] = 0 \text{ or } v_t[n+1] = 0)$ for all $t \neq t_0$. By the hypothesis, that yields $\kappa(p) = 0$, $X_{t_0} \in s(p)$ or $X_{t_0} \notin p$, $\kappa(q_{t_0}) \leq 1$, and $(X_t \notin p \text{ or } \kappa(q_t) = 0)$ for all $t \neq t_0$. Hence $\kappa(r) \leq 1$ by definition of r .

Case $w[j] = 0$. We have $\sum_{t=1}^n u[t] \bullet v_t[j] = 0$, implying $u[t] = 0$ or $v_t[j] = 0$ for all t . By the hypothesis, we obtain $X_t \notin p$ or $X_j \notin q_t$ for all t , implying $X_j \notin r$ as required.

Case $w[j] = 1$. We have $\sum_{t=1}^n u[t] \bullet v_t[j] = 1$, implying $u[t_0] = 1 = v_{t_0}[j]$ for some t_0 , and $(u[t] = 0 \text{ or } v_t[j] = 0)$ for all $t \neq t_0$. The hypothesis yields $X_{t_0} \in s(p)$ or $X_{t_0} \notin p$, $X_j \in s(q_{t_0})$ or $X_j \notin q_{t_0}$, and $(X_t \notin p \text{ or } X_j \notin q_t)$ for all $t \neq t_0$. That implies $X_j \in s(r)$ or $X_j \notin r$ as required. \square

Preparing the construction of certificates for loop statements, suppose that Y is a certificate for the body Q of a loop statement. Clearly, by Lemma 3.9 the k -fold iteration of Q , that is, $Q^k := Q; \dots; Q$ (k -times) for $k \geq 1$, has certificate Y^k , where, as usual, for $M \in \mathcal{M}_n[A]$ we write M^l for the l th iterate of M , that is, $M^0 := 1_n$ and $M^{l+1} := M \otimes M^l$.

Therefore, to obtain a certificate for all k -fold iterations of Q , it is natural to look at the limit Y^+ of all Y^k as defined below. Obviously, Y^+ is finite and efficiently computable. However, in order to verify the then constructed certificate for the given loop, we also consider the monotonic variant \hat{Y}^* of Y^+ .

DEFINITION 3.10 (limit forms). For $M \in \mathcal{M}_n[A]$, the limit forms M^+, M^* , and \hat{M}^* are

$$M^+ := \bigsqcup_{k=1}^{\infty} M^k, \quad M^* := 1_n \sqcup M^+, \quad \text{and} \quad \hat{M}^* := \bigsqcup_{k=0}^{\infty} \hat{M}^k, \quad \text{where} \quad \hat{M} := 1_n \sqcup M.$$

Observe that if M is a matrix in $\tilde{\mathcal{M}}_n[A]$, then so are M^+, M^* , and \hat{M}^* .

At some point in the construction of certificates for loop statements, we will proceed by induction on the partial ordering induced by the control of the certificate for the body of a given loop. According to the two forms of loop statements, there are two versions of the following partial ordering lemma.

DEFINITION 3.11 (control of M). For $M \in \mathcal{M}_n[A]$, the binary relation \rightarrow_M on

$\{1, \dots, n\}$, where $j \rightarrow_M i$ reads j controls i in M , is defined by

$$j \rightarrow_M i : \iff M[i][j] \geq 1.$$

Of course, given a certificate Z for a program P , $j \rightarrow_Z i$ means that the variable X_j may occur in $p_i \in \text{POLY}(Z[i])$ for a polynomial bound \vec{p} on P , and in that sense may control X_i in Z .

LEMMA 3.12 (partial ordering I). *Let M be in $\mathcal{M}_n[A]$ such that $\infty \notin \text{Diag}(\hat{M}^*)$. Then $\rightarrow_{\hat{M}^*}$ is a partial ordering of $\{1, \dots, n\}$.*

Proof. Reflexivity follows from $\hat{M}^* = \bigsqcup_{m \geq 0} (1_n \sqcup M)^m$. As for transitivity, suppose that $j \rightarrow_{\hat{M}^*} i$ and $i \rightarrow_{\hat{M}^*} k$. Then there exist $a, b \geq 0$ such that $\hat{M}^a[i][j] \geq 1$ and $\hat{M}^b[k][i] \geq 1$. We obtain $j \rightarrow_{\hat{M}^*} k$ as follows:

$$\begin{aligned} \hat{M}^* [k][j] &\geq \hat{M}^{b+a} [k][j] = (\hat{M}^b \otimes \hat{M}^a) [k][j] \\ &= \hat{M}^b [k][i] \bullet \hat{M}^a [i][j] + \sum_{d \neq i} \hat{M}^b [k][d] \bullet \hat{M}^a [d][j] \\ &\geq 1 \bullet 1 = 1. \end{aligned}$$

As for antisymmetry, we argue indirectly and assume $j \rightarrow_{\hat{M}^*} i$, $i \rightarrow_{\hat{M}^*} j$, and $i \neq j$. Thus, $\hat{M}^a [i][j] \geq 1$ and $\hat{M}^b [j][i] \geq 1$ for some $a, b \in \mathbb{N}$. As $i \neq j$, we know $a, b \geq 1$, since, e.g., $a = 0$ implied $\hat{M}^a [i][j] = 1_n [i][j] = 0$. By reflexivity of $\rightarrow_{\hat{M}^*}$, $\hat{M}^m [k][k] = (1_n \sqcup M)^m [k][k] \geq 1$ for any k , and $\infty \notin \text{Diag}(\hat{M}^*)$, we obtained as above the following contradiction:

$$1 = \hat{M}^* [j][j] \geq \hat{M}^b [j][i] \bullet \hat{M}^a [i][j] + \hat{M}^b [j][j] \bullet \hat{M}^a [j][j] \geq 1 + 1 = \infty. \quad \square$$

LEMMA 3.13 (partial ordering II). *Let M be any matrix in $\mathcal{M}_n[A]$ such that*

$$M^+ [i] = 1_n [i] \text{ or } M^+ [i][i] = 0 \text{ for } i = 1, \dots, n.$$

Then \rightarrow_{M^} is a partial ordering of $\{1, \dots, n\}$.*

Proof. Reflexivity of \rightarrow_{M^*} follows from $M^* = 1_n \sqcup M^+$, and transitivity follows as above, since $j \rightarrow_{M^*} i$ implies $M^a [i][j] \geq 1$ for some $a \geq 0$.

For antisymmetry, we argue indirectly and assume $j \rightarrow_{M^*} i$, $i \rightarrow_{M^*} j$, and $i \neq j$. As above, there exist $a, b \geq 1$ such that $M^a [i][j] \geq 1$ and $M^b [j][i] \geq 1$. That implied

$$M^+ [j][j] \geq (M^b \otimes M^a) [j][j] \geq M^b [j][i] \bullet M^a [i][j] \geq 1,$$

and thus $M^+ [j][j] \geq 1$, $M^+ [j][i] \geq 1$ with $i \neq j$, contradicting the hypothesis on M^+ . \square

Note that each $q \in \text{POLY}(\vec{a})$ is a polynomial in those variables X_i for which $a_i \geq 1$.

LEMMA 3.14 (certificate for loopI). *Let $P \equiv \text{loopI } X_h [Q]$ be any program, and let Y be a certificate for Q such that $\infty \notin \text{Diag}(\hat{Y}^*)$. Then the matrix Z with rows*

$$(z_1 \sqcup 1_{n+1} [1]), \dots, (z_n \sqcup 1_{n+1} [n]), 0^n 1$$

is a certificate for P , where z_1, \dots, z_n are defined as follows:

$$z_i := \begin{cases} Y^+ [i] & \text{if } \infty \notin Y^+ [i] \quad (\text{variable assignment}), \\ Y^+ [i] & \text{if } Y^+ [i] = 0^n \infty \quad (\text{constant assignment}), \\ 1_{n+1} [i] \sqcup 0^{h-1} Y [i][n+1] 0^{n+1-h} & \text{if } Y^+ [i] = 0^{i-1} 10^{n-i} \infty \quad (\text{push/inc}), \\ \text{else}(z_i) & \text{else} \quad (\text{else}), \end{cases}$$

where for $i \neq h$ (case $i = h$ falls under variable assignment) $\text{else}(z_i)$ is defined by

$$\text{else}(z_i)[j] := \begin{cases} \infty \bullet \hat{Y}^*[i][j] & \text{if } j \neq i, h, n+1, \\ 1 & \text{if } j = i, \\ \infty & \text{if } j = h, \\ 0 & \text{if } j = n+1. \end{cases}$$

Comments. The parts $\sqcup 1_{n+1}[i]$ in the i th row of Z take into account that P is not executed. Furthermore, the names of the first three cases in the above definition of z_i are not meant to cover the general situation but only refer to typical ones. In the variable assignment case, there exists a bounding polynomial on $|X_i|$ with simple variable occurrences only. As for the constant assignment case, there exists a constant bound on $|X_i|$. The push/inc case deals with the situation where each execution of Q increases $|X_i|$ by a constant ≥ 1 . In the else case, the idea is that because of $\infty \notin \text{Diag}(\hat{Y}^*)$ there exists a bounding polynomial of the form $X_i + X_h \cdot q(X_1, \dots, X_{i-1}, X_{i+1}, \dots, X_n)$, resulting in the definition of $\text{else}(z_i)$.

Proof. By Lemma 3.12 we can define a strict partial ordering of $\{1, \dots, n\}$ by setting

$$i \sqsubset_{\hat{Y}^*} j \iff i \rightarrow_{\hat{Y}^*} j \text{ and } i \neq j.$$

Thus, preparing the *else case* for z_i , we first proceed by induction on $\sqsubset_{\hat{Y}^*}$ showing the following.

CLAIM. *There exist polynomials $q_1(m, \vec{X}), \dots, q_n(m, \vec{X})$ such that for $i = 1, \dots, n$,*

- (A) $\{|\vec{X}| = \vec{s}\} Q^m \{|X_i| \leq s_i + q_i(m, \vec{s})\}$ for all $m \geq 0$.
- (B) $X_i + q_i(X_h, \vec{X}) \in \text{POLY}(\infty \bullet \hat{Y}^*[i] \sqcup H)$, where $H := 0^{h-1} \infty 0^{n+1-h}$.
- (C) $X_i \notin q_i$ and $\kappa(q_i) = 0$.

Proof of the claim. Consider any $i \in \{1, \dots, n\}$, and let i_1, \dots, i_l be those $i_j \neq i$ for which $i_j \rightarrow_{\hat{Y}^*} i$ holds. As Y is a certificate for Q with $\infty \notin \text{Diag}(\hat{Y}^*)$, there exists a polynomial $X_i + q'_i \in \text{POLY}(\hat{Y}[i])$ such that $X_i \notin q'_i$ and

$$(1) \quad \{|\vec{X}| = \vec{s}\} Q \{|X_i| \leq s_i + q'_i(\vec{s})\}.$$

Subcase $l = 0$. We have that $Y[i] \leq \hat{Y}^*[i] \leq 0^{i-1} 10^{n-i} \infty$; hence q'_i is a constant. Thus,

$$q_i(m, \vec{X}) := m \cdot q'_i$$

yields a polynomial clearly satisfying (C) and (A), the latter by (1), and (B) holds because $X_i + X_h \cdot q'_i \in \text{POLY}(1_{n+1}[i] \sqcup H) \subseteq \text{POLY}(\infty \bullet \hat{Y}^*[i] \sqcup H)$ by Lemma 3.7.

Subcase $l \geq 1$. We have that $i \neq h$, because the requirement on loop statements and Lemma 4.5 imply that Q is nonsize-increasing w.r.t. X_h , that is, $Y[h] \leq 1_{n+1}[h]$, implying $\hat{Y}^*[h] = 1_{n+1}[h]$. Now, the induction hypothesis yields polynomials r_{i_1}, \dots, r_{i_l} such that for $j = 1, \dots, l$,

- (IHA) $\{|\vec{X}| = \vec{s}\} Q^m \{|X_{i_j}| \leq s_{i_j} + r_{i_j}(m, \vec{s})\}$ for all $m \geq 0$.
- (IHB) $X_{i_j} + r_{i_j}(X_h, \vec{X}) \in \text{POLY}(\infty \bullet \hat{Y}^*[i_j] \sqcup H)$.
- (IHC) $X_{i_j} \notin r_{i_j}$ and $\kappa(r_{i_j}) = 0$.

Let j_1, \dots, j_k be those indices $j_s \neq i$ for which $j_s \rightarrow_Y i$ holds. As $Y[i] \leq \hat{Y}^*[i]$, all of those indices are among i_1, \dots, i_l . As q'_i is a polynomial in X_{j_1}, \dots, X_{j_k} , we set

$$q_i(m, \vec{X}) := m \cdot q'_i(\dots, X_{i_j} + r_{i_j}(m, \vec{X}), \dots),$$

where $X_{i_j} + r_{i_j}(m, \vec{X})$ is substituted for X_{i_j} just in case i_j is among j_1, \dots, j_k .

As for (C), clearly $\kappa(q_i) = 0$. To see $\mathbf{X}_i \notin q_i$, we argue indirectly, assuming $\mathbf{X}_i \in q_i$. Then $\mathbf{X}_i \in r_{i_j}$ for some i_j ; hence $\mathbf{X}_i \in \text{POLY}(\infty \bullet \hat{Y}^*[i_j] \sqcup H)$ by (IHB). As $i_j \neq i$ and $i \neq h$, that implied $i \rightarrow_{\hat{Y}^*} i_j \rightarrow_{\hat{Y}^*} i$. Thus, $i = i_j$ by antisymmetry of $\rightarrow_{\hat{Y}^*}$, contradicting $i \notin \{i_1, \dots, i_l\}$.

As for the proof of (A), we proceed by induction on $m \geq 0$, where *base case* $m = 0$ is obvious. For the *step case* $m \rightarrow m + 1$, let \vec{u} be the output size of $\vec{\mathbf{X}}$ after m rounds of \mathbf{Q} on input of size \vec{s} , and let \vec{v} be the output size of $\vec{\mathbf{X}}$ after another round of \mathbf{Q} on input of size \vec{u} . Then, by using monotonicity of polynomials, we conclude the step case as follows:

$$\begin{aligned} v_i &\leq u_i + q'_i(\vec{u}) && \text{by (1),} \\ &\leq s_i + q_i(m, \vec{s}) + q'_i(\dots, s_{i_j} + r_{i_j}(m, \vec{s}), \dots) && \text{by the I.H. for } m \text{ and (IHA),} \\ &\leq s_i + (m + 1) \cdot q'_i(\dots, s_{i_j} + r_{i_j}(m + 1, \vec{s}), \dots) && \text{by definition of } q_i(m, \vec{s}), \\ &= s_i + q_i(m + 1, \vec{s}). \end{aligned}$$

As for (B), we know $q'_i \in \hat{Y}[i]$, and $\mathbf{X}_{i_j} + r_{i_j}(\mathbf{X}_h, \vec{\mathbf{X}}) \in \text{POLY}(\infty \bullet \hat{Y}^*[i_j] \sqcup H)$ by (IHB). As the zero polynomial belongs to each class $\text{POLY}(\vec{a})$, Lemma 3.9 yields $r \in \text{POLY}(w)$, where $w := \hat{Y}[i] \otimes (\infty \bullet \hat{Y}^* \sqcup \hat{H})$, \hat{H} denotes the matrix in $\tilde{\mathcal{M}}_n[A]$ with rows $H, \dots, H, 0^{n-1}1$ (recall $H = 0^{h-1}10^{n+1-h}$), and

$$r(\mathbf{X}_h, \vec{\mathbf{X}}) := q'_i(\dots, \mathbf{X}_{i_j} + r_{i_j}(\mathbf{X}_h, \vec{\mathbf{X}}), \dots).$$

Thus, for the proof of (B), it suffices to show $(*) \text{POLY}(w) \subseteq \text{POLY}(\infty \bullet \hat{Y}^*[i] \sqcup H)$, because then (B) follows from $\mathbf{X}_i + q_i(\mathbf{X}_h, \vec{\mathbf{X}}) = \mathbf{X}_i + \mathbf{X}_h \cdot r(\mathbf{X}_h, \vec{\mathbf{X}})$, and $\mathbf{X}_i \notin q_i$, $i \neq h$. To see $(*)$, we argue using Lemma 3.7. First, observe that $\hat{Y}^* = \hat{Y}^e = \hat{Y}^{e+1}$ for some $e \geq 0$, implying $\hat{Y}[i] \otimes (\infty \bullet \hat{Y}^*) = \infty \bullet (\hat{Y}[i] \otimes \hat{Y}^*) = \infty \bullet \hat{Y}^*[i]$. Now consider any $j \in \{1, \dots, n+1\}$. If $j \neq h$, then $w[j] = (\hat{Y}[i] \otimes (\infty \bullet \hat{Y}^*)) [j] = \infty \bullet \hat{Y}^*[i][j] = (\infty \bullet \hat{Y}^*[i] \sqcup H)[j]$. Otherwise, if $j = h$, then $(\infty \bullet \hat{Y}^*[i] \sqcup H)[j] = \infty \bullet \hat{Y}^*[i][j] \sqcup \infty = \infty \geq w[j]$. Thus, in either case, $w[j] \leq (\infty \bullet \hat{Y}^*[i] \sqcup H)[j]$ as required. This completes the proof of the above claim. \square

Turning to the proof of the lemma, recall that we must prove the existence of polynomials $p_1 \in \text{POLY}(Z_1), \dots, p_n \in \text{POLY}(Z_n)$ such that

$$(2) \quad \{|\vec{\mathbf{X}}| = \vec{s}\} \mathbf{P} \{|\mathbf{X}_i| \leq p_i(\vec{s})\} \quad \text{for } i = 1, \dots, n.$$

Consider any $i \in \{1, \dots, n\}$. As Y is a certificate for \mathbf{Q} , there exists a polynomial $q_i \in \text{POLY}(Y[i])$ such that

$$(3) \quad \{|\vec{\mathbf{X}}| = \vec{s}\} \mathbf{Q} \{|\mathbf{X}_i| \leq q_i(\vec{s})\}.$$

According to the definition of z_i , we consider four cases.

Variable assignment case $\infty \notin Y^+[i]$. We have that $z_i = Y^+[i]$, and we conclude from (3) and Lemma 3.7 that $q_{z_i} \in \text{POLY}(z_i)$ satisfies $\{|\vec{\mathbf{X}}| = \vec{s}\} \mathbf{Q}^m \{|\mathbf{X}_i| \leq q_{z_i}(\vec{s})\}$ for all $m \geq 1$. Thus, $p_i := \mathbf{X}_i \sqcup q_{z_i}$ is a polynomial in $\text{POLY}(Z[i])$ such that (2) is true of p_i .

Constant assignment case $Y^+[i] = 0^n \infty$. We have that $z_i = Y^+[i]$, and by (3) there exists a constant polynomial c in $\text{POLY}(z_i)$ such that $\{|\vec{\mathbf{X}}| = \vec{s}\} \mathbf{Q} \{|\mathbf{X}_i| \leq c\}$. Thus, $p_i := \mathbf{X}_i + c$ is a polynomial in $\text{POLY}(Z[i])$ which satisfies (2).

Push/inc case $Y^+[i] = 0^{i-1}10^{n-i} \infty$. We have that $z_i = 1_{n+1}[i] \sqcup 0^{h-1}Y[i][n+1]0^{n+1-h}$, and by (3) and Lemma 3.7 there exists a polynomial $\mathbf{X}_i + b \in \text{POLY}(Y^+[i])$

such that $\{|\vec{X}| = \vec{s}\} \mathbf{Q} \{|\mathbf{X}_i| \leq s_i + b\}$. We conclude that $\{|\vec{X}| = \vec{s}\} \mathbf{Q}^m \{|\mathbf{X}_i| \leq s_i + b \cdot m\}$ for $m \geq 0$. Thus, $p_i := \mathbf{X}_i + b \cdot \mathbf{X}_h$ is a polynomial in $\text{POLY}(Z_i)$ for which (2) holds.

Else case $\infty \in Y^+[i] \neq 0^n \infty$ and $Y^+[i] \neq 0^{i-1}10^{n-i} \infty$. Then $i \neq h$, because the requirement on loop statements and Lemma 4.5 imply that \mathbf{Q} is nonsize-increasing w.r.t. \mathbf{X}_h , that is, $Y[h] \leq 1_{n+1}[h]$; hence $Y^+[h] \leq 1_{n+1}[h]$ and $\infty \notin Y^+[h]$. Thus, the case $i=h$ is appropriately treated in the variable assignment case. Now, let q_i be the polynomial obtained from the above Claim, satisfying (A), (B), (C). The current case is then completed by setting

$$p_i(\vec{X}) := \mathbf{X}_i + q_i(\mathbf{X}_h, \vec{X}).$$

Obviously, (2) is true of p_i by (A). To see $p_i \in \text{POLY}(Z[i]) = \text{POLY}(\text{else}(z_i))$, we argue using Corollary 3.3. If $Z[i][j] = 1$, then $j = i$; hence $\mathbf{X}_i \in s(p_i)$ by (C) and $i \neq h$. If $Z[i][j] = 0$, then $j \neq i, h$, and as $\kappa(p_i) = 0$, it suffices to consider the case $j \in \{1, \dots, n\} \setminus \{i, h\}$. Then $0 = Z[i][j] = \infty \bullet \hat{Y}^*[i][j] = (\infty \bullet \hat{Y}^*[i] \sqcup H)[j]$; hence (B) implies $\mathbf{X}_j \notin p_i$ as required. \square

LEMMA 3.15 (certificate for loopII). *Let $\mathbf{P} \equiv \text{loopII } \mathbf{X}_h \mathbf{Q}$ be any program, and let Y be a certificate for \mathbf{Q} satisfying*

$$(I) \quad Y^+[i] = 1_{n+1}[i] \text{ or } Y^+[i][i] = 0 \text{ for } i = 1, \dots, n + 1.$$

Then Y^ is a certificate for \mathbf{P} .*

Proof. First, we define polynomials $q_{1,m}, \dots, q_{n,m}$ for $m \geq 0$ s.t. for $i = 1, \dots, n$,

(A) $\{|\vec{X}| = \vec{s}\} \mathbf{Q}^m \{|\mathbf{X}_i| \leq q_{i,m}(\vec{s})\}$ for all $m \geq 0$.

(B) $q_{i,m} \in \text{POLY}(Y^m[i])$.

(C) There exists an $m_i \geq 0$ such that $\bigsqcup_{s \leq m} q_{i,s} \leq \bigsqcup_{s \leq m_i} q_{i,s}$ for all $m \geq 0$.

The proof that Y^* is a certificate for \mathbf{P} is then completed by setting (for $i = 1, \dots, n$)

$$p_i := \bigsqcup_{s \leq m_i} q_{i,s}$$

since (B) and Lemmas 3.8(a) and 3.7(a) imply $p_i \in \text{POLY}(\bigsqcup_{s \leq m_i} Y^s[i]) \subseteq \text{POLY}(Y^*[i])$, and from (A), (C) we obtain $\{|\vec{X}| = \vec{s}\} \mathbf{P} \{|\mathbf{X}_i| \leq q_{i,2^{\lceil s_h \rceil}}(\vec{s}) \leq p_i(\vec{s})\}$ as required.

As for the proof of (A) and (B), consider any $i \in \{1, \dots, n\}$. Since Y is a certificate for \mathbf{Q} , there exist polynomials $q'_1 \in \text{POLY}(Y[1]), \dots, q'_n \in \text{POLY}(Y[n])$ such that

$$(4) \quad \{|\vec{X}| = \vec{s}\} \mathbf{Q} \{|\mathbf{X}_j| \leq q'_j(\vec{s})\} \quad \text{and} \quad q'_i = \mathbf{X}_i \text{ whenever } Y[i] = 1_{n+1}[i]$$

for $j = 1, \dots, n$. Let i_1, \dots, i_l be those i_j such that $i_j \rightarrow_{Y^+} i$, and let $j_1 < \dots < j_k$ be those indices j_s such that $j_s \rightarrow_Y i$. Then $j_1, \dots, j_k \in \{i_1, \dots, i_l\}$, as $Y[i] \leq Y^+[i]$. We inductively define $q_{i,m}$ by

$$q_{i,0} := \mathbf{X}_i, \\ q_{i,m+1} := q'_i(r_1, \dots, r_n), \quad \text{where } r_{j_s} := q_{j_s,m} \text{ for } s = 1, \dots, k, \text{ and } r_t := 0 \text{ else.}$$

One easily verifies (A) by induction on $m \geq 0$, using (4). To see that (B) is true of $q_{i,m}$, observe that $q_{i,0} \in \text{POLY}(Y^0[i])$, and inductively $r_1, \dots, r_n \in \text{POLY}(Y^m[i])$, the hypothesis $q'_i \in \text{POLY}(Y[i])$, and Lemma 3.9(a) imply $q_{i,m+1} \in \text{POLY}(Y \otimes Y^m[i]) = \text{POLY}(Y^{m+1}[i])$.

For (C), we associate to each $q_{i,m}$ a *control tree* $\text{CT}_{i,m}$ with root (ℓ) and labels i, \perp :

$$\begin{aligned} \text{CT}_{i,0} &:= (i), \\ \text{CT}_{i,m+1} &:= \begin{cases} (\perp) & \text{if } l = 0 \ (Y^+[i] \in \{0\}^n A), \\ (i) & \text{if } l \geq 1, \ i \in \{j_1, \dots, j_k\} \ (Y^+[i] = 1_{n+1}[i]), \\ \langle (i), \text{CT}_{j_1,m}, \dots, \text{CT}_{j_k,m} \rangle & \text{if } l \geq 1, \ i \notin \{j_1, \dots, j_k\}. \end{cases} \end{aligned}$$

By Lemma 3.13, \rightarrow_{Y^*} is a partial ordering of $\{1, \dots, n\}$, and $Y \leq Y^+$. Therefore, for every path in any $\text{CT}_{i,m}$ from the root to a leaf, each label can occur at most once. Hence each such path is of length at most n , implying that there are only finitely many control trees. Thus, to prove (C), it suffices to show that the mapping $q_{i,m} \mapsto \text{CT}_{i,m}$ is injective, that is, for all $m, m' \geq 0$, and for $i = 1, \dots, n$,

(D) if $\text{CT}_{i,m} = \text{CT}_{i,m'}$ then $q_{i,m} = q_{i,m'}$.

The proof is by induction on the *depth* N of $\text{CT}_{i,m}$, assuming (H) $\text{CT}_{i,m} = \text{CT}_{i,m'}$.

Case $N = 0$. We have that $\text{CT}_{i,m} \in \{(i), (\perp)\}$. If $\text{CT}_{i,m} = (i)$, then $q_{i,m} = \mathbf{X}_i = q_{i,m'}$, because, e.g., $q_{i,0} = \mathbf{X}_i$, and $m > 0$ implies $k = 1$, $j_1 = i$, $r_i = q_{i,m-1} = \mathbf{X}_i$, and $q'_i = \mathbf{X}_i$; hence $q_{i,m} = \mathbf{X}_i$ by construction. Otherwise if $\text{CT}_{i,m} = (\perp)$, then $m, m' > 0$, and q'_i is a constant, implying $q_{i,m} = c = q_{i,m'}$ by construction.

Case $N > 0$. We have that (H) implies $m, m' > 0$ and $\text{CT}_{j_s,m-1} = \text{CT}_{j_s,m'-1}$ for $s = 1, \dots, k$. Thus, the I.H. yields $q_{j_s,m-1} = q_{j_s,m'-1}$ for $s = 1, \dots, k$, implying $q_{i,m} = q_{i,m'}$. \square

Now that we know how to construct certificates for psb basic instructions (see Note 3.5) and for composed programs (see Lemmas 3.8, 3.9, 3.14, 3.15), all ingredients are at hand to set up the *method* of certifying the psb property for the class of programs under consideration.

More precisely, we will define a *certification method* which assigns to each program P a value $M(P)$, which is either the *undefined value* \perp or a certificate $M(P)$ for P . As usual, we write $M(P) \downarrow$ for $M(P) \neq \perp$. Furthermore, we say that a program P is *certified* iff $M(P) \downarrow$.

DEFINITION 3.16 (certification method). *For any program P with variables among $\vec{X} = X_1, \dots, X_l$, $M(P)$ is defined inductively as follows:*

- $M(\text{imp}(\vec{X})) := \langle \text{imp}(\vec{X}), \vec{p} \rangle$ for psb basic instructions $\text{imp}(\vec{X})$ with polynomial bound \vec{p} .
- If $M(P_1) \downarrow$ and $M(P_2) \downarrow$, then $M(P_1; P_2) := M(P_2) \otimes M(P_1)$.
- If $M(P_1) \downarrow$ and $M(P_2) \downarrow$, then $M(\text{if}(\text{cond}) \text{ then } P_1 \text{ else } P_2)$ is defined as $M(P_1) \sqcup M(P_2)$.
- If $M(Q) \downarrow$, then $M(\text{if top}(X_k) \equiv a \text{ then } Q) := M(Q) \sqcup 1_{n+1}$.
- If P is $\text{loopI } X_h [Q]$ and $Y := M(Q)$ is a certificate for Q with $\infty \notin \text{Diag}(\hat{Y}^*)$, then $M(P) := Z$ as defined in Lemma 3.14.
- If P is $\text{loopII } X_h [Q]$ and $Y := M(Q)$ is a certificate for Q such that $Y^+[i] = 1_{n+1}[i]$ or $Y^+[i][i] = 0$ for $i = 1, \dots, n+1$, then $M(P) := Y^*$.
- In all other cases of P , $M(P) := \perp$.

THEOREM 3.17 (soundness). *For every program P with psb basic instructions only, if $M(P) \downarrow$, then $M(P)$ is a certificate for P ; in other words, P is psb, too.*

Proof. Following the cases of $M(P) \downarrow$, the statement of the theorem follows from Note 3.5 and Lemmas 3.8, 3.9, 3.14, and 3.15. \square

For reasons similar to those elaborated on in [16], it is an undecidable problem whether a given program is psb. As a consequence, there is no limit to refining the above certification method so as to recognize more and more psb programs. We give three examples.

Suppose that P is a sequence $P_1; P_2$ such that P_2 is a loop with body Q and control variable X_h , and $M(P_1) \downarrow$. If $M(P_1)[h] = 0^{n+1}$, then Q will never be executed; thus

P_2 can be ignored when computing $M(P)$. Furthermore, if $M(P)[h] = 0^n 1$, then Q is executed at most once. In that case, we obtain $(M(Q) \sqcup 1_{n+1}) \otimes M(P_1)$ as a certificate for P whenever $M(Q) \downarrow$.

Moreover, for nonsize-increasing basic instructions (cf. Definition 4.4) one obtains a certificate $\leq 1_{n+1}$. Those instructions can be ignored when computing $M(P)$. However, it would be wiser to keep basic instructions like $\text{nil}(X)$, for they can lead to better bounding polynomials.

4. Embedding stack and loop programs of μ -measure 0. In this section, we will show that all stack and loop programs of μ -measure 0 [16] are certified by the present method. That embedding will be used only to facilitate the proof of the characterization theorems below. The strength of the present method, however, is not based on the measure μ . First we review stack programs and the measure μ .

Stack programs operate on stacks X, Y, Z, \dots over a fixed but arbitrary alphabet. Such programs are built from the usual basic instructions $\text{push}(a, X)$, $\text{pop}(X)$, $\text{nil}(X)$ by sequencing $P_1; P_2$, conditional statements $\text{if } \text{top}(X) \equiv a \text{ then } Q$, and loopI statements $\text{foreach } X [Q]$ (read *for each symbol in X do Q*) provided that no push , pop , or nil instruction with respect to the control stack X occurs in Q . The operational semantics of stack programs is standard, except possibly for loop statements $\text{foreach } X [Q]$: they are executed *call-by-value* such that during the execution, every symbol in X can be inspected while preserving its contents.

Central to the design of the measure μ is the notion of control.

DEFINITION 4.1 (control). *Let P be any stack program, and let $\text{PUSH}(P)$ be the set of all Y appearing as $\text{push}(a, Y)$ in P . The control of P is the transitive closure \rightarrow_P of the following governance relation \rightarrow_P :*

- X governs Y in P ($X \rightarrow_P Y$) if P contains a loop statement $\text{foreach } X [R]$ such that $Y \in \text{PUSH}(R)$.

Thus, X controls Y in P ($X \rightarrow_P Y$) iff $X \rightarrow_P Z_1 \rightarrow_P \dots \rightarrow_P Z_l \rightarrow_P Y$ for some Z_1, \dots, Z_l .

Observe that $Z \rightarrow_Q Z$ is precluded by the syntactic restrictions on loop statements.

DEFINITION 4.2 (measure μ). *For stack programs P , the μ -measure of P , denoted by $\mu(P)$, is inductively defined (and streamlined) as follows:*

- $\mu(\text{imp}(\vec{X})) := 0$ for every basic instruction $\text{imp}(\vec{X})$.
- $\mu(\text{if } \text{top}(X) \equiv a \text{ then } Q) := \mu(Q)$.
- $\mu(P_1; P_2) := \max\{\mu(P_1), \mu(P_2)\}$.
- $\mu(\text{foreach } X [Q]) := \begin{cases} \mu(Q) + 1 & \text{if } Q \text{ has a top circle,} \\ \mu(Q) & \text{else,} \end{cases}$

where Q has a top circle if it contains a loop $T := \text{foreach } Y [R]$ such that $\mu(T) = \mu(Q)$ and $Z \rightarrow_Q Y$ for some $Z \in \text{PUSH}(R)$.

One obtains $\mu(\text{foreach } X [\text{foreach } X [Q]]) = \mu(\text{foreach } X [Q])$, and furthermore, if Q is a sequence $Q_1; \dots; Q_k$ with a top circle, then some component Q_i contains a loop $T := \text{foreach } Y [R]$ such that $\mu(T) = \mu(Q)$ and $Z \rightarrow_Q Y$ for some $Z \in \text{PUSH}(R)$. The latter clearly shows that this measure μ admits more programs at any higher level than the original version in [16] does, for it requires the existence of a component Q_i such that $\mu(Q_i) = \mu(Q)$ and $U \rightarrow_{Q_i} V$ and $V \rightarrow_{Q^{-i}} U$ for some U, V , where Q^{-i} is Q without component Q_i .

The measure μ on loop programs is defined in the same way, except that the size-increasing instruction $\text{suc}(X)$ plays the role of the push operation.

Loop programs are built from the basic instructions $\text{suc}(X)$ (increment the number stored in X by one), $\text{nil}(X)$ (set X to zero), and $\text{pred}(X)$ (decrement the number stored in X by one) by sequencing $P_1; P_2$ and loopI statements $\text{loop } X [Q]$ (execute Q x times

whenever x is stored in \mathbf{X}), provided that no instruction $\text{succ}(\mathbf{X})$, $\text{nil}(\mathbf{X})$, or $\text{pred}(\mathbf{X})$ occurs in \mathbf{Q} . Here variables serve as registers, storing natural numbers.

As shown in [16], stack programs of μ -measure 0 compute precisely the functions in FPTIME , and loop programs of μ -measure 0 exactly compute the functions of the Grzegorzczuk class \mathcal{E}_2 which is identical to the class FLINSPACE of the linear-space computable functions [32].

To show that $\mu(\mathbf{P})=0$ implies $M(\mathbf{P}) \downarrow$, we need to embed the analysis of control inherent in the measure μ in the certification method. Some technical lemmata are needed beforehand.

LEMMA 4.3 (iteration). *Let M be in $\tilde{\mathcal{M}}_n[A]$ such that $j \rightarrow_{M^m} i$ for some $m \geq 1$, where $i, j \in \{1, \dots, n\}$. Then there exist $i_1 = j, i_2, \dots, i_k = i \in \{1, \dots, n\}$ such that*

$$j = i_1 \rightarrow_M i_2 \rightarrow_M \dots \rightarrow_M i_k = i$$

and i_1, \dots, i_k are pairwise distinct, except for i_1, i_k .

Proof. We proceed by induction on $m \geq 1$. The base case $m = 1$ holds by the hypothesis on M . For the step case $m \rightarrow m + 1$, note that $M[n + 1] = 0^n 1$ implies

$$M^{m+1}[i][j] = \sum_{d=1}^n M[i][d] \bullet M^m[d][j],$$

and hence $M[i][e], M^m[e][j] \geq 1$ for some $e \in \{1, \dots, n\}$ by the hypothesis on M^{m+1} . The I.H. yields $i_1 = j, i_2, \dots, i_k = e$ such that $j = i_1 \rightarrow_M i_2 \rightarrow_M \dots \rightarrow_M i_k = e$, and i_1, \dots, i_k are pairwise distinct, except for i_1, i_k .

Subcase $j \neq e$. If $i \notin \{i_1, \dots, i_k\}$, then we are done, as $e \rightarrow_M i$. Otherwise let l be minimal such that $i_l = i$. If $j \neq i$, we are done; otherwise $i \rightarrow_M i_2 \rightarrow_M \dots \rightarrow_M i_k \rightarrow_M i$ will do.

Subcase $j = e$. In this case, we are done, since $e \rightarrow_M i$. \square

DEFINITION 4.4. *A program \mathbf{P} with certificate $M(\mathbf{P})$ is nonsize-increasing w.r.t. \mathbf{X}_i iff $M(\mathbf{P})[i] \leq 1_{n+1}[i]$.*

Obviously, $\text{pop}(\mathbf{X}_i)$ and $\text{nil}(\mathbf{X}_i)$ are nonsize-increasing w.r.t. \mathbf{X}_i (cf. section 5).

LEMMA 4.5 (nonsize-increasing). *If a certified program \mathbf{P} contains only nonsize-increasing basic instructions w.r.t. \mathbf{X}_i , then \mathbf{P} is also nonsize-increasing w.r.t. \mathbf{X}_i .*

Proof. The proof is by induction on the structure of \mathbf{P} with $Z = M(\mathbf{P})$. The case where \mathbf{P} is a basic instruction follows from the hypothesis on \mathbf{P} .

Case $\mathbf{P} \equiv \mathbf{P}_1; \mathbf{P}_2$. Then $Z = Y_2 \otimes Y_1$, where $Y_k = M(\mathbf{P}_k)$ for $k = 1, 2$, and the induction hypothesis yields $Y_k[i] \leq 1_{n+1}[i]$ for $k = 1, 2$. Thus, for $j = 1, \dots, n + 1$ we obtain

$$Z[i][j] = (Y_2 \otimes Y_1)[i][j] = \sum_{d=1}^{n+1} Y_2[i][d] \bullet Y_1[d][j] \leq Y_1[i][j] \leq 1_{n+1}[i][j].$$

Case $\mathbf{P} \equiv \text{if}(\text{cond}) \text{ then } \mathbf{P}_1 \text{ else } \mathbf{P}_2$. Then $Z = Y_1 \sqcup Y_2$, where $Y_k = M(\mathbf{P}_k)$ for $k = 1, 2$. The I.H. yields $Y_k[i] \leq 1_{n+1}[i]$ for $k = 1, 2$, and hence $Z[i] \leq 1_{n+1}[i]$.

Case $\mathbf{P} \equiv \text{loopI } \mathbf{X}_h[\mathbf{Q}]$. Then $\infty \notin \text{Diag}(\hat{Y}^*)$ for $Y = M(\mathbf{Q})$, and $Y[i] \leq 1_{n+1}[i]$ by the I.H. on \mathbf{Q} . We conclude $\hat{Y}[i] = 1_{n+1}[i]$; hence $Y^+[i] \leq 1_{n+1}[i]$ and $Z[i] = Y^+[i] \sqcup 1_{n+1}[i] = 1_{n+1}[i]$ as required. Note that $Z[i]$ falls under the variable assignment case of Lemma 3.14.

Case $\mathbf{P} \equiv \text{loopII } \mathbf{X}_h[\mathbf{Q}]$. Then $Z = Y^*$, where $Y = M(\mathbf{Q})$, and $Y[i] \leq 1_{n+1}[i]$ by the I.H. on \mathbf{Q} . From the above we conclude that $Y^+[i] \leq 1_{n+1}[i]$; hence $Z[i] \leq 1_{n+1}[i]$. \square

LEMMA 4.6 (control embedding). *For any stack or loop program P with certificate $Z = M(P)$, and for all $i, j \in \{1, \dots, n\}$,*

- *if $Z[i][j] \geq 1$ and $i \neq j$, then $X_j \rightarrow_P X_i$;*
- *if $Z[i][i] = \infty$, then $X_i \rightarrow_P X_i$.*

NOTE. *To see that the two cases above are distinct, consider the stack program $P \equiv \text{foreach } X_h [\text{pop}(X_i)]$. As $M(\text{pop}(X_i)) = 1_{n+1}$, we obtain $M(P) = 1_{n+1}$, in particular $M(P)[i][i] = 1$, but $\neg(X_i \rightarrow_P X_i)$.*

Proof. The proof is by induction on the structure of P with $Z = M(P)$. We treat only stack programs, as the proof for loop programs is almost identical. The case where P is a basic instruction holds by pure logic, since the hypothesis fails for $M(P)$.

Case P is if $\text{top}(X_k) \equiv a$ then Q. Then $Z = M(Q) \sqcup 1_{n+1}$, and the claim follows from the I.H., as $Z[i][j] = M(Q)[i][j]$ for $i \neq j$, and $Z[i][i] = \infty$ implies $M(Q)[i][i] = \infty$.

Case $P \equiv P_1; P_2$. Then $Z = Y_2 \otimes Y_1$, where $Y_k = M(P_k)$. As $Y_1[n+1][j] = 0$, we get

$$(*) \quad Z[i][j] = \sum_{d=1}^n Y_2[i][d] \bullet Y_1[d][j].$$

First consider the case where $Z[i][j] \geq 1$ and $i \neq j$. Then (*) implies $Y_2[i][e], Y_1[e][j] \geq 1$ for some $e \in \{1, \dots, n\}$. Then either i, j, e are distinct or $e \in \{i, j\}$. In either case, the I.H. yields $X_e \rightarrow_{P_2} X_i$ and $X_j \rightarrow_{P_1} X_e$, implying $X_j \rightarrow_P X_i$ as required.

Consider the case $Z[i][i] = \infty$. By (*) for $i = j$, we can consider two subcases.

Subcase $Y_2[i][e] \bullet Y_1[e][i] = \infty$ for some $e \in \{1, \dots, n\}$. Then each factor is ≥ 1 , and one is ∞ . If $e = i$, then $X_i \rightarrow_P X_i$ follows from the I.H. on an ∞ factor. Otherwise the I.H. yields $X_e \rightarrow_{P_2} X_i$ and $X_i \rightarrow_{P_1} X_e$, implying $X_i \rightarrow_P X_i$ as required.

Subcase $Y_2[i][e] \bullet Y_1[e][i] = 1$ and $Y_2[i][d] \bullet Y_1[d][i] = 1$ for some $e \neq d$. We may assume $i \neq e$. As each factor is 1, the I.H. yields $X_e \rightarrow_{P_2} X_i$ and $X_i \rightarrow_{P_1} X_e$. This implies $X_i \rightarrow_P X_i$, concluding the current case $P \equiv P_1; P_2$.

Case $P \equiv \text{foreach } X_h [Q]$. Suppose that $Y = M(Q)$. Then case $Z[i][i] = \infty$ is ruled out by construction and $\infty \notin \text{Diag}(\hat{Y}^*)$. Suppose that $Z[i][j] \geq 1$ and $i \neq j$. We consider several subcases according to the definition of $Z = M(P)$.

Subcase $\infty \notin Y^+[i]$ or $Y^+[i] = 0^n \infty$. Then $Z[i] = Y^*[i]$, and $Z[i][j] \geq 1$ implies $Y^m[i][j] \geq 1$ for some $m \geq 1$, as $i \neq j$. Lemma 4.3 yields pairwise distinct $j = i_1, i_2, \dots, i_k = i$ such that

$$j = i_1 \rightarrow_Y i_2 \rightarrow_Y \dots \rightarrow_Y i_k = i.$$

Therefore, the I.H. yields $X_{i_1} \rightarrow_Q X_{i_2} \rightarrow_Q \dots \rightarrow_Q X_{i_k}$, implying $X_j \rightarrow_P X_i$.

Subcase $Y^+[i] = 0^{i-1}10^{n-i}\infty$. Then $Z[i] = 1_{n+1}[i] \sqcup 0^{h-1}Y[i][n+1]0^{n+1-h}$, and $i \neq j$ and $Z[i][j] \geq 1$ imply $j = h$ and $Y[i][n+1] \geq 1$. Thus, we must show $X_h \rightarrow_P X_i$. It suffices to show that Q contains a push operation on X_i . If Q contained no such operation, then Q would be nonsize-increasing w.r.t. X_i by Lemma 4.5; hence $Y[i] \leq 1_{n+1}[i]$. But that implied $Y^+[i] \leq 1_{n+1}[i]$, contradicting the subcase assumption.

Subcase $\infty \in Y^+[i]$ and $Y^+[i] \neq 0^n \infty, 0^{i-1}10^{n-i}\infty$. Now $i \neq j$ and $Z[i][j] \geq 1$ imply $j = h$ or $\hat{Y}^*[i][j] \geq 1$. If $j = h$ then, as $\infty \in Y^+[i]$, we argue as above that Q contains a push operation on X_i , implying $X_h \rightarrow_P X_i$. Otherwise $\hat{Y}^*[i][j] \geq 1$ and

$i \neq j$ imply $\hat{Y}^m[i][j] \geq 1$ for some $m \geq 1$; hence Lemma 4.3 yields pairwise distinct $j = i_1, i_2, \dots, i_k = i$ such that

$$j = i_1 \rightarrow_{\hat{Y}} i_2 \rightarrow_{\hat{Y}} \dots \rightarrow_{\hat{Y}} i_k = i.$$

Now observe that $\hat{Y}[d][e] \geq 1$ and $d \neq e$ imply $Y[d][e] \geq 1$. Thus, the I.H. yields $X_{i_1} \rightarrow_{\mathbb{Q}} X_{i_2} \rightarrow_{\mathbb{Q}} \dots \rightarrow_{\mathbb{Q}} X_{i_k}$, implying $X_j \rightarrow_{\mathbb{P}} X_i$ as required. \square

THEOREM 4.7 (embedding). *All stack/loop programs of μ -measure 0 are certified.*

Proof. The proof is by induction on the structure of \mathbb{P} with $\mu(\mathbb{P}) = 0$. We treat only stack programs, as the proof for loop programs is almost identical. The case where \mathbb{P} is a basic instruction `push`, `pop`, or `nil` is obvious.

Case \mathbb{P} is `if top(X_k) \equiv a then \mathbb{Q} .` Then $\mu(\mathbb{Q}) = 0$, and the I.H. yields $M(\mathbb{P}) = M(\mathbb{Q}) \sqcup 1_{n+1}$.

Case $\mathbb{P} \equiv \mathbb{P}_1; \mathbb{P}_2$. Then $\mu(\mathbb{P}_1) = \mu(\mathbb{P}_2) = 0$, and the I.H. yields $M(\mathbb{P}) = M(\mathbb{P}_2) \otimes M(\mathbb{P}_1)$.

Case $\mathbb{P} \equiv \text{foreach } X_h \text{ [} \mathbb{Q} \text{].}$ Then $\mu(\mathbb{Q}) = 0$ and \mathbb{Q} contains no (top) circle; that is, \mathbb{Q} contains no variable X_i such that $X_i \rightarrow_{\mathbb{Q}} X_i$. By the I.H., $Y := M(\mathbb{Q})$ is defined, and we must show $\infty \notin \text{Diag}(\hat{Y}^*)$. We argue indirectly and assume $\hat{Y}^*[i][i] = \infty$ for some $i \in \{1, \dots, n\}$. Note that $i \neq h$, since \mathbb{Q} contains no `push` operation on the control stack X_h of \mathbb{P} , and hence \mathbb{Q} is nonsize-increasing w.r.t. X_h by Lemma 4.5.

From the assumption, we obtain $\hat{Y}^m[i][i] = \infty$ for some $m \geq 1$. We show by induction on $m \geq 1$ that this implies $X_i \rightarrow_{\mathbb{Q}} X_i$, contradicting $\mu(\mathbb{P}) = 0$. In the *base case* $m = 1$, $\hat{Y}[i][i] = \infty$ implies $Y[i][i] = \infty$, and hence $X_i \rightarrow_{\mathbb{Q}} X_i$ by Lemma 4.6.

For the *step case* $m \rightarrow m+1$, observe that $\hat{Y}[n+1] = 0^n 1$ and $i \in \{1, \dots, n\}$ imply

$$\hat{Y}^{m+1}[i][i] = \sum_{d=1}^n \hat{Y}[i][d] \bullet \hat{Y}^m[d][i] = \infty.$$

Subcase $\hat{Y}[i][e] \bullet \hat{Y}^m[e][i] = \infty$ for some $e \in \{1, \dots, n\}$. If $e = i$ and $\hat{Y}[i][i] = \infty$, then $Y[i][i] = \infty$, implying $X_i \rightarrow_{\mathbb{Q}} X_i$ by Lemma 4.6. If $e = i$ and $\hat{Y}^m[i][i] = \infty$, then $X_i \rightarrow_{\mathbb{Q}} X_i$ follows from the induction hypothesis. Otherwise $e \neq i$ and $\hat{Y}[i][e], \hat{Y}^m[e][i] \geq 1$. In that case, Lemmas 4.3 and 4.6 imply $X_e \rightarrow_{\mathbb{Q}} X_i$ and $X_i \rightarrow_{\mathbb{Q}} X_e$; hence $X_i \rightarrow_{\mathbb{Q}} X_i$.

Subcase $\hat{Y}[i][e] \bullet \hat{Y}^m[e][i] = 1$ and $\hat{Y}[i][d] \bullet \hat{Y}^m[d][i] = 1$ for some $e \neq d$. In that case, we may assume $e \neq i$, and we argue as in the previous subcase to show $X_i \rightarrow_{\mathbb{Q}} X_i$. \square

5. String programs and FPTIME. This section is concerned with showing that certified “string programs” exactly compute the functions in FPTIME.

DEFINITION 5.1 (string programs). *A basic instruction $\text{imp}(\vec{X})$ is admissible if it can be simulated on a Turing machine in polynomial time.*

String programs are stack programs that have arbitrary admissible basic instructions and are extended by the following conditional with expected operational semantics:

$$\text{if top}(X_i) \equiv a \text{ then } \mathbb{P}_1 \text{ else } \mathbb{P}_2.$$

Observe that admissible basic instructions are psb.

Examples. The following are examples of admissible basic instructions, where w is any word over Σ :

- The *no operation*: $\{\vec{X} = \vec{v}\} \text{nop } \{\vec{X} = \vec{v}\}$.
- The *swap operation*: $\{X_i, X_j = v_i, v_j\} \text{swap}(X_i, X_j) \{X_i, X_j = v_j, v_i\}$.

- The *constant assignment statement*: $\{X_i = v_i\} X_i = w \{X_i = w\}$.
- The *assignment statement*: $\{X_i, X_j = v_i, v_j\} X_i = X_j \{X_i, X_j = v_j, v_j\}$.
- The *constant concatenation*: $\{X_i = v_i\} X_i += w \{X_i = v_i w\}$.
- The *concatenation statement*: $\{X_i, X_j = v_i, v_j\} X_i += X_j \{X_i, X_j = v_i v_j, v_j\}$.

We use the following notation for simple certificates.

DEFINITION 5.2 ($Z_{i,j,a}$, $A_{i,j,a}$, and F). For programs in variables X_1, \dots, X_n , let

- $Z_{i,j,a}$ result from 1_{n+1} by replacing row i with $0^{j-1} a 0^{n+1-j}$, and
- $A_{i,j,a}$ result from 1_{n+1} by replacing row i with $1_{n+1}[i] + 0^{j-1} a 0^{n+1-j}$.

The forgetting function $F: \mathbb{N} \rightarrow A$ is defined by $F(0) := 0$, $F(1) := 1$, $F(n+2) := \infty$.

Certificates. The following are certificates for some admissible basic instructions with variables among X_1, \dots, X_n :

- 1_{n+1} is a certificate for both $\text{pop}(X_i)$ and nop .
- One obtains a certificate $S_{i,j}$ for $\text{swap}(X_i, X_j)$ by swapping in 1_{n+1} row i with row j .
- Using the notation in Definition 5.2,

$Z_{i,i,0}$	is a certificate for $\text{nil}(X_i)$,
$Z_{i,n+1,F(w)}$	is a certificate for $X_i = w$,
$Z_{i,j,1}$	is a certificate for $X_i = X_j$,
$A_{i,n+1,1}$	is a certificate for $\text{push}(a, X_i)$,
$A_{i,n+1,F(w)}$	is a certificate for $X_i += w$,
$A_{i,j,1}$	is a certificate for $X_i += X_j$.

Observe that, in the presence of the conditionals, $\text{if } \text{top}(X_i) \equiv a \text{ then } P_1 \text{ else } P_2$ and nop , conditionals of the form $\text{if } \text{top}(X_i) \equiv a \text{ then } P_1$ can be dispensed with, because they can be defined by $\text{if } \text{top}(X_i) \equiv a \text{ then } P_1 \text{ else nop}$ with an identical certificate, if any.

DEFINITION 5.3 (sharp form). For $X \in \tilde{\mathcal{M}}_n[A]$ and any choice of $*_1, \dots, *_{n+1} \in \{0, 1, \infty\}$, a sharp form $X^\#$ of X is a matrix in $\tilde{\mathcal{M}}_{n+1}[A]$ defined as follows:

$$X = \begin{pmatrix} X_1 & \cdots & X_n & X_{n+1} \\ 0 & \cdots & 0 & 1 \end{pmatrix} \implies X^\# = \begin{pmatrix} X_1 & \cdots & X_n & \mathbf{0} & X_{n+1} \\ *_1 & \cdots & *_n & 1 & *_{n+1} \\ 0 & \cdots & 0 & 0 & 1 \end{pmatrix}.$$

DEFINITION 5.4. An accumulator of X_1, \dots, X_n is any string program in variables X_1, \dots, X_n, X_{n+1} such that $C \in \tilde{\mathcal{M}}_{n+1}[A]$ is a certificate for that program, where C is obtained from 1_{n+2} by replacing row $n+1$ with $1^{n+1}0$.

Note that the certificate of an accumulator of X_1, \dots, X_n is a sharp form of 1_{n+1} . The following program ADD is a typical example of an accumulator of X_1, \dots, X_n :

$$\text{ADD} := X_{n+1} += X_1; \dots; X_{n+1} += X_n.$$

LEMMA 5.5 (sharp forms and accumulators). Let ACC be any accumulator of $\vec{X} := X_1, \dots, X_n$, and let P be any string program in variables \vec{X} such that $M(P) \downarrow$.

- If $X, Y \in \tilde{\mathcal{M}}_n[A]$, then $Y^\# \otimes X^\#$ is a sharp form $(Y \otimes X)^\#$, and $Y^\# \sqcup X^\#$ is a sharp form $(Y \sqcup X)^\#$.
- $M(P; \text{ACC})$ is a sharp form $M(P)^\#$.
- If $P^\#$ is obtained from P by replacing each admissible basic instruction $\text{imp}(\vec{X})$ with the sequence $\text{imp}(\vec{X}); \text{ACC}$, then $M(P^\#)$ is a sharp form $M(P)^\#$.

Proof. Part (a) holds by pure matrix multiplication. Part (b) holds by pure matrix multiplication together with the fact that if $Y \in \tilde{\mathcal{M}}_n[A]$ is a certificate for P ,

then

$$Y' = \begin{pmatrix} Y_1 \cdots Y_n \mathbf{0} Y_{n+1} \\ 0 \cdots 0 \ 1 \ 0 \\ 0 \cdots 0 \ 0 \ 1 \end{pmatrix}$$

is a certificate for P as a program with variables among \vec{X}, X_{n+1} . The proof of (c) is by induction on the structure of P , where the case $P \equiv \text{imp}(\vec{X})$ follows from (b).

If P is a sequence $P_1; P_2$ or a conditional $\text{if } \text{top}(X_i) \equiv a \text{ then } P_1 \text{ else } P_2$, then $P^\#$ is $P_1^\#; P_2^\#$ or $\text{if } \text{top}(X_i) \equiv a \text{ then } P_1^\# \text{ else } P_2^\#$, and the claim follows from the I.H., part (a), and Definition 3.16.

Consider the case $P \equiv \text{foreach } X_h [Q]$. Then $P^\# \equiv \text{foreach } X_h [Q^\#]$, and by the I.H. $M(Q^\#)$ is a sharp form $Y^\#$ of $Y := M(Q)$. Furthermore, as $\infty \notin \text{Diag}(\hat{Y}^*)$, and as $\widehat{Y^\#}^*$ is a sharp form $(\hat{Y}^*)^\#$ by (a), we conclude that $\infty \notin \text{Diag}(\widehat{Y^\#}^*)$, too. Thus, $M(P^\#)$ is defined, and inspecting all cases in the construction of $M(P^\#)$ in Lemma 3.14, we see that $M(P^\#)$ is a sharp form $M(P)^\#$. \square

THEOREM 5.6 (characterization of FPTIME). *Certified string programs exactly compute the functions in FPTIME.*

Proof. For the implication $\boxed{\Leftarrow}$, consider any f in FPTIME. By [16] and Theorem 4.7, f can be computed by a stack program of μ -measure 0 such that $M(P) \downarrow$.

As for $\boxed{\Rightarrow}$, let P be a certified string program with variables among $\vec{X} := X_1, \dots, X_n$. Then let $\text{TIME}_P(\vec{w})$ denote the number of steps in a run of P on input \vec{w} , where a *step* is the execution of any admissible basic instruction $\text{imp}(\vec{X})$. Observe that there is a polynomial $q_{\text{time}}(\vec{n})$ such that each step $\text{imp}(\vec{X})$ can be simulated by a Turing machine in time $q_{\text{time}}(|\vec{X}|)$. Then let $P^\#$ result from P by replacing each basic instruction $\text{imp}(\vec{X})$ with the sequence $\text{imp}(\vec{X}); \text{ADD}$, where ADD is the accumulator of \vec{X} defined above.

By Lemma 5.5 we obtain that $M(P^\#)$ is a sharp form $M(P)^\#$. Thus, by soundness (Theorem 3.17) there exists a polynomial $q \in \text{POLY}(M(P^\#)[n+1])$ such that

$$\{\vec{X}, X_{n+1} = \vec{w}, v\} P^\# \{ |X_{n+1}| \leq q(|\vec{w}|, |v|) \}.$$

By the construction of $P^\#$ we see that $\text{TIME}(P) := \text{nil}(X_{n+1}); \text{ADD}; P^\#$ satisfies

$$\{\vec{X} = \vec{w}\} \text{TIME}(P) \{ |X_{n+1}| \geq \text{TIME}_P(\vec{w}) \}$$

and that $p(|\vec{w}|) := q(|\vec{w}|, \sum_i |w_i|)$ bounds the size of each stack at any time during the execution of P on input \vec{w} . Thus, every step $\text{imp}(\vec{X})$ in a run of P on input \vec{w} can be simulated on a Turing machine in time $q_{\text{time}}(p(|\vec{w}|), \dots, p(|\vec{w}|))$. Referring to standard simulations of string programs on Turing machines (e.g., cf. section 7), the existence of a Turing machine which simulates P on input \vec{w} in time $O(p(|\vec{w}|) \cdot q_{\text{time}}(p(|\vec{w}|), \dots, p(|\vec{w}|)))$ follows. \square

6. General loop programs and FLINSPACE. In this section we will show that certified “general loop programs” exactly compute the functions in FLINSPACE.

DEFINITION 6.1 (general loop programs). *A basic instruction $\text{imp}(\vec{X})$ is admissible if it can be simulated on a Turing machine in linear space.*

General loop programs are loop programs that have arbitrary admissible basic instructions and are extended by the following conditional with expected operational semantics:

$$\text{if } X_i \leq X_j \text{ then } P_1 \text{ else } P_2.$$

Examples. The following are examples of admissible basic instructions, where p is any polynomial in \vec{X} :

- the basic instructions $\text{nil}(X_i)$, $\text{pred}(X_i)$, $\text{suc}(X_i)$ of loop programs, just as nop and $\text{swap}(X_i, X_j)$, bearing in mind that now \vec{X} serve as *registers*.
- the *assignment statement* $X_i = p$ satisfying $\{\vec{X} = \vec{x}\} X_i = p \{X_i = p(\vec{x})\}$.
- the *increase statement* $X_i += p$ satisfying $\{\vec{X} = \vec{x}\} X_i += p \{X_i = x_i + p(\vec{x})\}$.

In particular, the instructions $X_i = c$ and $X_i = X_j$, as well as $X_i += c$ and $X_i += X_j$, are admissible basic instructions, where c is any constant.

Certificates. The following are certificates for some admissible basic instructions with variables among X_1, \dots, X_n :

- The certificates for $\text{nil}(X_i)$, $\text{pred}(X_i)$, $\text{suc}(X_i)$, nop , and $\text{swap}(X_i, X_j)$ are just those for $\text{nil}(X_i)$, $\text{pop}(X_i)$, $\text{push}(a, X_i)$, nop , and $\text{swap}(X_i, X_j)$ as given in section 6.
- One obtains a certificate for $X_i = p$ from 1_{n+1} by replacing row i with $\langle p \rangle$.
- By replacing in 1_{n+1} row i with $1_{n+1}[i] + \langle p \rangle$, one obtains a certificate for $X_i += p$.

THEOREM 6.2 (characterization of FLINSPACE). *Certified general loop programs exactly compute the functions in FLINSPACE.*

Proof. For the implication $\boxed{\Leftarrow}$, consider any f in FLINSPACE. By [16] and Theorem 4.7, f can be computed by a loop program of μ -measure 0 such that $M(P) \downarrow$.

As for $\boxed{\Rightarrow}$, let P be any certified general loop program. By soundness (Theorem 3.17), every f computed by P has a polynomial bound. Thus, by closure of \mathcal{E}_2 under bounded simultaneous primitive recursion, we obtain inductively that every f computed by P is in $\mathcal{E}_2 = \text{FLINSPACE}$. \square

7. Power string programs and FPSPACE. This section concerns an extension of string programs we call power string programs such that certified power string programs exactly compute the functions in FPSPACE.

DEFINITION 7.1 (power string programs). *A basic instruction is ps-admissible if it can be simulated on a Turing machine in polynomial space.*

Power string programs are string programs that are built from ps-admissible basic instructions and are extended by the following loopII statement we call power loop:

$$\text{repeat Pow}(X_h) [Q],$$

which executes the body $2^{|w|}-1$ times whenever w is initially stored in the control stack X_h .

An example of a ps-admissible basic instruction used in the proof of the characterization theorem above is the *truncate instruction* $\text{truncate}(X_i, X_j)$ satisfying

$$\{X_i, X_j = w_i, w_j\} \text{truncate}(X_i, X_j) \{X_j = w_j \text{ and } X_i = \text{trunc}(w_i, w_j)\},$$

where $\text{trunc}(u, v) := u$ if $|u| \leq |v|$, and $\text{trunc}(u, v) := u'$ if $u = u'u''$ and $|u'| = |v|$.

Using the notation of Definition 5.2, we obtain $Z_{i,j,1}$ as a certificate for $\text{truncate}(X_i, X_j)$.

If the method certifies a power string program P , then every function computed by P will be polynomially size bounded. However, when simulating P on a Turing machine $\text{sim}(P)$, the space used by $\text{sim}(P)$ might exceed the polynomial bounds provided by $M(P)$. Therefore, to prove that power string programs with certificate $M(P)$ can be simulated on a Turing machine $\text{sim}(P)$ in polynomial space, we first *standardize* those simulations.

Given a power string program P with variables among X_1, \dots, X_n , the simulation $\text{sim}(P)$ has an *input/output tape*, a *working tape* $T(X_i)$ for each X_i , and a *reference tape* $R(X_i)$ for each occurrence of a loop controlled by X_i .

Every run of $\text{sim}(\text{P})$ on input $w_{i_1} \# \dots \# w_{i_n}$ is divided into three phases, where phase 1 or 3 can be skipped, depending on the use of $\text{sim}(\text{P})$ as a subprogram of another program: the *initialization* phase, in which each component w_{i_j} is copied to the corresponding working tape $T(\mathbf{X}_{i_j})$, the *step-by-step simulation* phase, and the *output preparation* phase.

All cases of $\text{sim}(\text{P})$ are obvious, except possibly the case where P is a loop statement **foreach** $\mathbf{X}_h[\text{Q}]$ or **repeat** $\text{Pow}(\mathbf{X}_h)[\text{Q}]$. Given $\text{sim}(\text{Q})$, let $\text{sim}(\text{Q})^*$ result from $\text{sim}(\text{Q})$ by modifying its transition function such that every move w.r.t. $T(\mathbf{X}_h)$ becomes a move w.r.t. $R(\mathbf{X}_h)$.

$\text{sim}(\text{foreach } \mathbf{X}_h[\text{Q}])$ works on input \vec{w} as follows: After the initialization phase, the contents of tape $T(\mathbf{X}_h)$ is copied to the reference tape $R(\mathbf{X}_h)$, leaving its tape head on the top symbol of \mathbf{X}_h . Then for each inspected symbol on $R(\mathbf{X}_h)$ distinct from B , first call $\text{sim}(\text{Q})^*$, then delete the actual “top symbol” of tape $R(\mathbf{X}_h)$, and move its tape head one cell to the left.

$\text{sim}(\text{repeat } \text{Pow}(\mathbf{X}_h)[\text{Q}])$ works on input \vec{w} as follows: After the initialization phase, write $y := |w_h|$ ones (the binary representation of $2^{|w_h|} - 1$) or $y := 0$ (in case of $w_h = \varepsilon$) on the reference tape $R(\mathbf{X}_h)$. Then in an obvious loop of $2^{|w_h|} - 1$ rounds, first call $\text{sim}(\text{Q})$, and then compute the new $y := \text{bin}((y)_2 - 1)$ on tape $R(\mathbf{X}_h)$ from the actual y .

THEOREM 7.2 (characterization of FPSPACE). *Certified power string programs exactly compute the functions in FPSPACE .*

Proof. As for $\boxed{\Rightarrow}$, we proceed by induction on the structure of certified power string programs P , showing $S_{\text{sim}(\text{P})}(n) \leq p(n)$ for some polynomial $p(n)$.

The base case where P is a ps-admissible basic instruction holds by the fact that if a power string program can be simulated on a Turing machine in polynomial space, then it can also be simulated on a standardized Turing machine in polynomial space.

The cases where P is a sequence or a conditional easily follow by the I.H. and closure of polynomials under composition.

*Remaining cases, where P is a loop statement **foreach** $\mathbf{X}_h[\text{Q}]$ or **repeat** $\text{Pow}(\mathbf{X}_h)[\text{Q}]$.* By the I.H. there exists a polynomial $q(n)$ such that $S_{\text{sim}(\text{Q})}(n) \leq q(n)$. Furthermore, since $M(\text{P})$ is a certificate for P , there exists a polynomial b , built from those polynomials obtained in Lemma 3.14 or Lemma 3.15, respectively, such that

$$\{s_1 = |\mathbf{X}_1|, \dots, s_n = |\mathbf{X}_n|\} \text{P} \{|\mathbf{X}_i| \leq b(\max\{s_1, \dots, s_n\})\} \text{ for } i = 1, \dots, n.$$

Then a straightforward induction on s_h shows that $S_{\text{sim}(\text{P})}(n) \leq (b \sqcup q \circ b)(n)$.

As for the proof of $\boxed{\Leftarrow}$, let f be any function in FPSPACE computed by a one-tape Turing machine $M := (Q, \Gamma, \Sigma, q_0, \delta)$ such that $S_M(n) \leq p(n)$ for some polynomial $p(n)$. As M halts on every input, one can find another polynomial $q(n)$ such that $T_M(n) \leq 2^{q(n)}$. Let b be the polynomial $p \sqcup q$. To obtain a certified power string program P that simulates M , we follow [16], [29] and modify the given stack program simulation of M as follows.

The required program P satisfying $\{\mathbf{X} = w\} \text{P} \{0 = f_M(w)\}$ uses stacks $\mathbf{X}, \mathbf{Y}, \mathbf{Z}, \mathbf{L}, \mathbf{R}, 0, \dots$, and is of the form

$$\begin{array}{ll} \text{P} ::= \text{COMPUTE-SPACE-BOUND}(\mathbf{Y}); & (* \text{ of } \mu\text{-measure } 0 *) \\ \text{INITIALIZE}(\mathbf{L}, \mathbf{Z}, \mathbf{R}); & (* \text{ of } \mu\text{-measure } 0 *) \\ \text{repeat Pow}(\mathbf{Y}) [\text{SIMULATE-MOVES}; \text{TRUNCATE-ALL}]; & (* ? *) \\ \text{OUTPUT}(\mathbf{R}; 0) & (* \text{ of } \mu\text{-measure } 0 *) \end{array}$$

where $\text{TRUNCATE-ALL} ::= \text{truncate}(\mathbf{X}_1, \mathbf{Y}); \dots; \text{truncate}(\mathbf{X}_n, \mathbf{Y})$, and $\vec{\mathbf{X}} = \text{Var}(\text{P})$. Let INIT be the initial part $\text{COMPUTE-SPACE-BOUND}(\mathbf{Y}); \text{INITIALIZE}(\mathbf{L}, \mathbf{Z}, \mathbf{R})$, and let Q be

the body of the power loop. Then the simulation of M is such that for each configuration $\alpha(\mathbf{q}, \mathbf{a})\beta$ obtained after m steps of M on w , that is, $\text{init}_M(w) \vdash^m \alpha(\mathbf{q}, \mathbf{a})\beta$, we have

$$\{\mathbf{X} = w\} \text{INIT}; \mathbf{Q}^m \{\mathbf{L} = \alpha, \text{reverse}(\mathbf{R}) = \mathbf{a}\beta, \mathbf{Z} = \mathbf{q}\}.$$

Observe that, apart from the use of the power loop and `truncate`, everything else in \mathbf{P} belongs to the stack program world. In particular, the stack program `COMPUTE-SPACE-BOUND(Y)` computes in stack \mathbf{Y} a word of size $b(|w|)$, bounding the space that M uses on input w . As well, `SIMULATE-MOVES` is a stack program of μ -measure 0. Hence by Theorem 4.7, all of the components `INIT`, `Q`, and `OUTPUT(R; 0)` are certified. Thus, to obtain a certificate $M(\mathbf{P})$, it remains to verify that the certificate for `Q` satisfies the hypothesis of Lemma 3.15, that is,

$$(I) \quad M(\mathbf{Q})^+[i] = 1_{n+1}[i] \text{ or } M(\mathbf{Q})^+[i][i] = 0 \text{ for } i = 1, \dots, n + 1.$$

First observe that `TRUNCATE-ALL` is the only place in \mathbf{P} where `truncate` is used, and where \mathbf{Y} appears in `Q`. Furthermore, by construction of the bound b , none of those truncate operations changes the contents of any \mathbf{X}_i . However, assuming $\mathbf{X}_n = \mathbf{Y}$ (w.l.o.g.), and bearing in mind that $Z_{i,n,1}$ is the certificate of `truncate(Xi, Xn)`, the effect of `TRUNCATE-ALL` is that

$$M(\mathbf{Q}) = \begin{pmatrix} 0 \cdots 010 \\ \cdots \\ 0 \cdots 010 \\ 0 \cdots 001 \end{pmatrix},$$

whatever the certificate of `SIMULATE-MOVES`. Thus (I) is true of $M(\mathbf{Q})$. \square

Inspecting again the certificate $M(\mathbf{Q})$ above, we see that for the proof of $\boxed{\Leftarrow}$, a trivialized hypothesis on the certificate of a loopII-body in Lemma 3.15 would have sufficed. But note again that the focus here is not on characterization theorems but on certifying as many programs as possible, and that is why the hypothesis (I) in Lemma 3.15 is so valuable.

8. Applications. In this section, examples of certified natural algorithms are given, some of which are considered benchmark algorithms [14] in implicit computational complexity. Furthermore, one algorithm is given where the certification method correctly fails.

8.1. Basic arithmetical operations. For legibility, we write

$$\text{prog}(\vec{\mathbf{X}}_1, \dots, \mathbf{X}_r; \mathbf{Y}_1, \dots, \mathbf{Y}_s)$$

for *subprograms*, where $\vec{\mathbf{X}}$ act as *read-only input parameters*, and $\vec{\mathbf{Y}}$ as *output parameters*. All variables of the subprogram not appearing among $\vec{\mathbf{X}}, \vec{\mathbf{Y}}$ are *local variables* and must be discharged at the end of the subprogram. That allows us to suppress in subprogram certificates all rows and columns corresponding to local variables.

The following programs are straightforward implementations of elementary methods of basic arithmetical operations, and it is natural to use stacks of bits here. Accordingly, we use the following two kinds of variables:

- variables `A, B, S, ...` of data type “`stack < bit >`” with `bit := {0, 1}`. As usual, the *size* of a stack is the length of the word stored in it.
- variables `carry, sum_bit` of data type `bit`. The *size* of a bit is 1.

In addition to those admissible basic instructions and their certificates as shown in section 5, we use the following admissible basic instructions, where \mathbf{b} , \mathbf{S} are aliases for X_i, X_j and $\langle \text{BE} \rangle$ is any polynomial-time computable Boolean expression:

$\mathbf{b} = \langle \text{BE} \rangle$	assigns the value of $\langle \text{BE} \rangle$ to \mathbf{b}	certificate: $Z_{i,n+1,1}$,
$\text{push}(\mathbf{b}, \mathbf{S})$	pushes the bit stored in \mathbf{b} on \mathbf{S}	certificate: $A_{j,n+1,1}$,
$\text{reverse}(\mathbf{S})$	reverses the word stored in \mathbf{S}	certificate: 1_{n+1} .

Alternatively, using Lemma 3.14 and the above convention of suppressing local variables, one could implement $\text{reverse}(\mathbf{S})$ with certificate 1_{n+1} .

BINARY ADDITION. This procedure uses $\mathbf{A}, \mathbf{B}, \mathbf{S}, \mathbf{A}', \mathbf{B}', \text{carry}$, and sum_bit as aliases for X_1, \dots, X_7 .

```

1: procedure BIN_ADD( $\mathbf{A}, \mathbf{B}; \mathbf{S}$ )
2:    $\mathbf{A}' = \mathbf{A}; \mathbf{B}' = \mathbf{B};$                                 ▷ Creates local copies of  $\mathbf{A}, \mathbf{B}$ 
3:   if ( $\text{len}(\mathbf{B}') < \text{len}(\mathbf{A}')$ ) then
4:      $\mathbf{S} = \mathbf{B}';$ 
5:      $\mathbf{B}' = \mathbf{A}';$ 
6:      $\mathbf{A}' = \mathbf{S};$                                         ▷ Sets the control variable  $\mathbf{B}'$ 
7:    $\text{nil}(\mathbf{S});$                                           ▷ Initializes the result stack
8:    $\text{carry} = 0;$                                         ▷ Initializes the carry
9:   foreach  $\mathbf{B}'$  do
10:     $\text{sum\_bit} = \text{top}(\mathbf{A}') \text{ xor } \text{top}(\mathbf{B}') \text{ xor } \text{carry};$ 
11:     $\text{push}(\text{sum\_bit}, \mathbf{S});$ 
12:     $\text{carry} = \text{top}(\mathbf{A}') \text{ and } \text{top}(\mathbf{B}');$ 
13:     $\text{pop}(\mathbf{A}');$ 
14:   if ( $\text{carry}$ ) then
15:      $\text{push}(1, \mathbf{S});$                                   ▷ Now  $\mathbf{S}$  holds the result string in reverse order
16:    $\text{reverse}(\mathbf{S});$ 
17:    $\text{nil}(\mathbf{A}'); \text{nil}(\mathbf{B}')$                             ▷ Discharges local copies

```

To compute the certificate, we refer to the following parts of the algorithm: P_1 for lines 2–8, P_2 for the loop with body Q , and P_3 for lines 14–17. For those parts, the method yields

$$\begin{aligned}
M(P_1) &= Z_{6,8,1} \otimes Z_{3,3,0} \otimes ((Z_{4,3,1} \otimes Z_{5,4,1} \otimes Z_{3,5,1}) \sqcup 1_8) \otimes Z_{5,2,1} \otimes Z_{4,1,1}, \\
M(P_3) &= Z_{5,5,0} \otimes Z_{4,4,0} \otimes 1_8 \otimes (A_{3,8,1} \sqcup 1_8), \\
Y &:= M(Q) = 1_8 \otimes Z_{6,8,1} \otimes A_{3,8,1} \otimes Z_{7,8,1}.
\end{aligned}$$

Thus, we obtain

$$M(P_1) = \begin{pmatrix} 10000000 \\ 01000000 \\ 00000000 \\ 11000000 \\ 11000000 \\ 00000001 \\ 00000010 \\ 00000001 \end{pmatrix}, \quad M(P_3) = \begin{pmatrix} 10000000 \\ 01000000 \\ 00100001 \\ 00000000 \\ 00000000 \\ 00000100 \\ 00000010 \\ 00000001 \end{pmatrix}, \quad \text{and } Y = \begin{pmatrix} 10000000 \\ 01000000 \\ 00100001 \\ 00010000 \\ 00001000 \\ 00000001 \\ 00000001 \\ 00000001 \end{pmatrix}.$$

Application of Lemma 3.14 yields the following certificate $M(P_2)$ for the loop:

$$\text{As } \hat{Y}^* = \begin{pmatrix} 1000000 0 \\ 0100000 0 \\ 0010000 \infty \\ 0001000 0 \\ 0000100 0 \\ 0000010 \infty \\ 0000001 \infty \\ 0000000 1 \end{pmatrix}, Y^+ = \begin{pmatrix} 1000000 0 \\ 0100000 0 \\ 0010000 \infty \\ 0001000 0 \\ 0000100 0 \\ 0000000 1 \\ 0000000 1 \\ 0000000 1 \end{pmatrix}, \text{ we get } M(P_2) = \begin{pmatrix} 1000000 0 \\ 0100000 0 \\ 0010\boxed{1}00\boxed{0} \\ 0001000 0 \\ 0000100 0 \\ 0000010 1 \\ 0000001 1 \\ 0000000 1 \end{pmatrix}.$$

Altogether, we obtain the following certificate for binary addition:

$$M(\text{bin_add}(A, B; S)) = M(P_3) \otimes M(P_2) \otimes M(P_1) = \begin{pmatrix} 1000000 0 \\ 0100000 0 \\ 1100000 1 \\ 0000000 0 \\ 0000000 0 \\ 0000000 \infty \\ 0000001 1 \\ 0000000 1 \end{pmatrix}.$$

BINARY MULTIPLICATION. This procedure uses $A, B, C,$ and tmp as aliases for $X_1, \dots, X_4.$

```

1: procedure BIN_MULT(A,B; C)
2:   nil(C);                                ▷ Initializes the result stack
3:   foreach B do
4:     push(0, C);                            ▷ Performs a shift-left of C
5:     if (top(B) ≡ 1) then
6:       bin_add(A, C; tmp);
7:       C = tmp;
8:       nil(tmp);

```

To compute the certificate, let Q denote the loop body (lines 4–8). As binary addition is called as a subprogram, we can use the certificate obtained above, but suppressing local variables and mapping arguments A, C, tmp to the parameters A, B, S of bin_add , that is,

$$M(\text{bin_add}(A, C; \text{tmp})) = \begin{pmatrix} 10000 \\ 01000 \\ 00100 \\ 10101 \\ 00001 \end{pmatrix}.$$

First, for the certificate $Y := M(Q)$ of the loop body Q we obtain

$$Y = ((Z_{4,4,0} \otimes Z_{3,4,1} \otimes M(\text{bin_add}(A, C; \text{tmp}))) \sqcup 1_5) \otimes A_{3,5,1}.$$

To obtain $M(\text{foreach } B[Q])$, we need to compute \hat{Y}^* and Y^+ . Since $Y = \hat{Y}$, this simplifies the calculations of the limit forms:

$$Y = \begin{pmatrix} 1000 0 \\ 0100 0 \\ 1010 \infty \\ 0001 0 \\ 0000 1 \end{pmatrix} \quad \text{and} \quad Y^+ = \hat{Y}^* = \begin{pmatrix} 1 000 0 \\ 0 100 0 \\ \infty 010 \infty \\ 0 001 0 \\ 0 000 1 \end{pmatrix}.$$

As $\infty \notin \text{Diag}(\hat{Y}^*)$, Lemma 3.14 can be applied, yielding

$$M(\text{foreach } B [Q]) = \begin{pmatrix} 1 & 0 & 000 \\ 0 & 1 & 000 \\ \infty & \infty & 100 \\ 0 & 0 & 010 \\ 0 & 0 & 001 \end{pmatrix}. \text{ Thus, } M(\text{bin_mult}(A, B; C)) = \begin{pmatrix} 1 & 0 & 000 \\ 0 & 1 & 000 \\ \infty & \infty & 000 \\ 0 & 0 & 010 \\ 0 & 0 & 001 \end{pmatrix}.$$

BINARY EXPONENTIATION. This procedure, which on input n computes 2^n , uses A, B, aux , and tmp as aliases for X_1, \dots, X_4 .

```

1: procedure BIN_EXP(A; B)
2:   B = 1;                                ▷ Initializes the result stack
3:   aux = 10;                              ▷ Initializes the auxiliary stack
4:   foreach A do
5:     bin_mult(aux, B; tmp);
6:     B = tmp

```

Clearly, as `bin_exp` cannot run in polynomial time, the certification method must fail. Let Q denote the loop body (lines 5–6). As `bin_mult` is called as a subprogram, we can use the certificate obtained above, again suppressing local variables and mapping arguments $\text{aux}, B, \text{tmp}$ to the parameters A, B, C of `bin_mult`:

$$M(\text{bin_mult}(\text{aux}, B; \text{tmp})) = \begin{pmatrix} 1 & 0 & 0 & 00 \\ 0 & 1 & 0 & 00 \\ 0 & 0 & 1 & 00 \\ 0 & \infty & \infty & 00 \\ 0 & 0 & 0 & 01 \end{pmatrix} \text{ and } Y := M(Q) = \begin{pmatrix} 1 & 0 & 0 & 00 \\ 0 & \infty & \infty & 00 \\ 0 & 0 & 1 & 00 \\ 0 & \infty & \infty & 00 \\ 0 & 0 & 0 & 01 \end{pmatrix}.$$

As $\infty \in \text{Diag}(\hat{Y}^*)$, Lemma 3.14 is not applicable; hence the method correctly fails.

8.2. Insertion-sort. The previous examples demonstrate that our certification method appropriately processes size-increasing programs; the following certification of insertion-sort exemplifies that the method applies to nonsize-increasing programs, too. To our knowledge, no implicit description of `FPTIME` includes a natural implementation of insertion-sort—Leivant’s first-order tiered systems typically involve needing a second copy of the list to be sorted; Hofmann’s [14] nonsize-increasing “coin” types require higher-type functionals. By contrast, our implementation, modeled after [10, p. 8] stays in the realm of imperative programming, and in that way, some nontrivial headway has been made.

For simplicity, the `INSERTION_SORT` algorithm sorts arrays of natural numbers. Accordingly, we use the following two kinds of variables:

- Variables i, j, k, \dots of data type “`Nat`” store natural numbers, with logarithmic *size measure* $|i| := \lceil \log_2(n + 1) \rceil$ whenever n is stored in i .
- Variables A, \dots of data type “`array(Nat)`” store sequences (of length $\text{len}(A)$) of natural numbers, with logarithmic *size measure*

$$|A| := \text{len}(A) \cdot \max_{1 \leq i \leq \text{len}(A)} \{|n_i| + 1\}$$

whenever $(n_1, \dots, n_{\text{len}(A)})$ is stored in A .

The implementation of `INSERTION_SORT` uses the following admissible basic instructions (cf. Definition 5.2 for the notation of certificates), where i, j, A are aliases for

X_i, X_j, X_k :

<code>dec(i)</code>	decrements any $n > 0$ stored in i by 1	certificate: 1_{n+1} ,
<code>inc(i)</code>	increments any n stored in i by 1	certificate: $A_{i,n+1,1}$,
<code>i = c</code>	assigns any constant $c \geq 0$ to i	certificate: $Z_{i,n+1,F(c)}$,
<code>i = j</code>	assigns the storage of j to i	certificate: $Z_{i,j,1}$,
<code>i = ones(len(A))</code>	assigns $2^{\text{len}(A)} - 1$ to i	certificate: $Z_{i,k,1}$,
<code>i = len(A)</code>	assigns $\text{len}(A)$ to i	certificate: $Z_{i,k,1}$,
<code>A[i] = A[j]</code>	assigns $A[j]$ to $A[i]$, $1 \leq i, j \leq \text{len}(A)$	certificate: 1_{n+1} .

For the certificate of `i = ones(len(A))`, observe that $|2^{\text{len}(A)} - 1| = \text{len}(A) \leq |A|$.

The INSERTION_SORT algorithm sorts an array A of type `array(Nat)`. As $A[1]$ is sorted, in each round the algorithm inserts the current *key* $A[i+1]$ into the sorted array $A[1 \dots i]$ (cf. [10, p. 8]). In the absence of while statements in the given framework, a natural implementation could appear as shown below, where INSERTION_SORT(A) sorts $A[1 \dots n]$, using $A[n+1]$ to buffer the current key.

```

1: procedure INSERTION_SORT(A)
2:   len = len(A);
3:   rounds = ones(len(A));
4:   j = 2;                                ▷ Prepares the for loop starting with 2
5:   loop rounds
6:     if (j < len) then
7:       i = j;
8:       dec(i);                            ▷ Prepares the downto loop starting with  $j - 1$ 
9:       A[len] = A[j];                    ▷ Buffers the current key  $A[j]$ 
10:      loop rounds
11:        if (i > 0 and A[i] > A[len]) then
12:          i_tmp = i;
13:          inc(i_tmp);
14:          A[i_tmp] = A[i];              ▷ Shifts  $A[i]$  one component to the right
15:          i_tmp = 0;
16:          dec(i);                          ▷ Prepares the next round of the downto loop
17:          i_tmp = i;                        ▷ Treats the cases  $\text{key} < A[1]$  or  $A[i] \leq \text{key}$ 
18:          inc(i_tmp);
19:          A[i_tmp] = A[len];
20:          i_tmp = 0;
21:          inc(j)                            ▷ Prepares the next round of the for loop

```

In order to compute the certificate for this algorithm, we refer to the variables A , len , rounds , j , i , and i_tmp as X_1, \dots, X_6 , respectively, and to the following parts of the algorithm:

P_1 \equiv the body `if (j < len) then [P2;P3;P4]` of the outer loop,
 P_2 \equiv `i = j ; dec(i); A[len]=A[j]`,
 P_3 \equiv the inner loop with body P_5 ,
 P_4 \equiv lines 17–21,
 P_5 \equiv the body of the conditional `if (i > 0 and A[i] > A[len]) then [P6]`.

According to the method, we obtain the following certificates:

$$\begin{aligned} M(\mathbf{P}_6) &= 1_7 \otimes Z_{6,7,0} \otimes 1_7 \otimes A_{6,7,1} \otimes Z_{6,5,1} = Z_{6,7,0}, \\ M(\mathbf{P}_5) &= M(\mathbf{P}_6) \sqcup 1_7 = 1_7, \\ M(\mathbf{P}_4) &= A_{4,7,1} \otimes Z_{6,7,0} \otimes 1_7 \otimes A_{6,7,1} \otimes Z_{6,5,1} = A_{4,7,1} \otimes Z_{6,7,0}, \\ M(\mathbf{P}_3) &= 1_7. \end{aligned}$$

$$\begin{aligned} M(\mathbf{P}_2) &= 1_7 \otimes 1_7 \otimes Z_{5,4,1} = Z_{5,4,1}, \\ Y := M(\mathbf{P}_1) &= (M(\mathbf{P}_4) \otimes M(\mathbf{P}_3) \otimes M(\mathbf{P}_2)) \sqcup 1_7 = A_{4,7,1} \sqcup A_{5,4,1}, \\ \hat{Y}^* &= \bigsqcup_{i=1}^{\infty} (\hat{Y})^i = A_{4,7,\infty} \sqcup A_{5,7,\infty} \sqcup A_{5,4,\infty}. \end{aligned}$$

As $\infty \notin \text{Diag}(\hat{Y}^*)$, Lemma 3.14 yields $M(\text{loopI rounds } [\mathbf{P}_1]) = A_{4,3,1} \sqcup A_{5,3,\infty} \sqcup A_{5,4,\infty}$. Thus, we obtain $M(\text{INSERTION_SORT}) = (A_{4,3,1} \sqcup A_{5,3,\infty} \sqcup A_{5,4,\infty}) \otimes Z_{4,7,\infty} \otimes Z_{3,1,1} \otimes Z_{2,1,1}$.

9. Conclusion. We have presented a new method of certifying polynomial size boundedness for imperative programs under the natural assumption that the basic instructions are polynomially size bounded, too. Apart from that, there are no restrictions on the data structures involved. We proved that certified string programs exactly compute the FPTIME functions, certified general loop programs precisely compute the FLINSPACE functions, and finally, that certified power string programs exactly compute the FPSPACE functions. We also gave examples of certified “natural” implementations of algorithms such as insertion-sort. Altogether, this can be considered a major step towards applicability of research in the evolving field of implicit computational complexity to daily programming practice. It seems that the future will see further work in this area.

Acknowledgments. We would like to thank Martin Dietzfelbinger for both his precious comments and suggestions concerning the presentation of the material. Along these lines, we are also grateful to the referees for their thorough reports. Furthermore, we are greatly indebted to Jan Mehler for pointing out various misprints in the paper, and in particular for his contribution to the proof of Lemma 3.15.

REFERENCES

- [1] K. AEHLIG AND J. JOHANNSEN, *An elementary fragment of second-order lambda calculus*, ACM Trans. Comput. Log., 6 (2005), pp. 468–480.
- [2] S. J. BELLANTONI, *Predicative Recursion and Computational Complexity*, Ph.D. thesis, University of Toronto, Toronto, Ontario, Canada, 1993.
- [3] S. J. BELLANTONI, *Predicative recursion and the polytime hierarchy*, in Feasible Mathematics II, P. Clote and J. Remmel, eds., Birkhäuser Boston, Boston, MA, 1995, pp. 15–29.
- [4] S. J. BELLANTONI AND S. COOK, *A new recursion-theoretic characterization of the polytime functions*, Comput. Complexity, 2 (1992), pp. 97–110.
- [5] S. J. BELLANTONI AND K.-H. NIGGL, *Ranking primitive recursions: The low Grzegorzcyk classes revisited*, SIAM J. Comput., 29 (1999), pp. 401–415.
- [6] S. J. BELLANTONI, K.-H. NIGGL, AND H. SCHWICHTENBERG, *Higher type recursion, ramification and polynomial time*, Ann. Pure Appl. Logic, 104 (2000), pp. 17–30.
- [7] G. BONFANTE, A. CICHON, J.-Y. MARION, AND H. TOUZET, *Algorithms with polynomial interpretation termination proof*, J. Funct. Programming, 11 (2001), pp. 33–53.
- [8] G. BONFANTE, A. CICHON, J.-Y. MARION, AND H. TOUZET, *Complexity classes and rewrite systems with polynomial interpretation*, in Computer Science Logic, Lecture Notes in Comput. Sci. 1584, Springer, Berlin, 1999, pp. 372–384.
- [9] P. CLOTE, *A safe recursion scheme for exponential time*, in Logical Foundations of Computer Science, Lecture Notes in Comput. Sci. 1234, Springer, Berlin, 1997, pp. 44–52.

- [10] T. H. CORMEN, C. E. LEISERSON, AND R. L. RIVEST, *Introduction to algorithms*, MIT Press, Cambridge, MA, 1990.
- [11] CRISS (Contrôle de Ressources et d'Interférence dans les Systèmes Synchrones). <http://www.cmi.univ-mrs.fr/~amadio/Criss/criss.html>.
- [12] A. GRZEGORCZYK, *Some classes of recursive functions*, *Rozprawy Mat.*, 4 (1953), pp. 1–45.
- [13] M. HOFMANN, *Type Systems for Polynomial-time Computation*, Habilitation Thesis, Technische Universität Darmstadt, Darmstadt, Germany, 1998.
- [14] M. HOFMANN, *Linear types and nonsize-increasing polynomial time computation*, *Inform. and Comput.*, 183 (2003), pp. 57–85.
- [15] N. D. JONES, *The expressive power of higher-order types or, life without CONS*, *J. Functional Programming*, 11 (2001), pp. 55–94.
- [16] L. KRISTIANSEN AND K.-H. NIGGL, *On the computational complexity of imperative programming languages*, *Theoret. Comput. Sci.*, 318 (2004), pp. 139–161.
- [17] L. KRISTIANSEN AND K.-H. NIGGL, *The Garland measure and computational complexity of stack programs*, *Electron. Notes Theoret. Comput. Sci.*, 90 (2003), pp. 15–35.
- [18] D. LEIVANT, *Subrecursion and lambda representation over free algebras*, in *Feasible Mathematics*, S. Buss and P. Scott, eds., Birkhäuser Boston, Boston, MA, 1990, pp. 281–291.
- [19] D. LEIVANT, *A foundational delineation of poly-time*, *Inform. and Comput.*, 110 (1994), pp. 391–420.
- [20] D. LEIVANT, *Stratified functional programs and computational complexity*, in *Conference Record of the Twentieth Annual ACM Symposium on Principles of Programming Languages*, ACM, New York, 1993, pp. 325–333.
- [21] D. LEIVANT, *Ramified recurrence and computational complexity I: Word recurrence and poly-time*, in *Feasible Mathematics II*, P. Clote and J. Remmel, eds., Birkhäuser Boston, Boston, MA, 1995, pp. 320–343.
- [22] D. LEIVANT, *Predicative recurrence in finite types*, in *Logical Foundations of Computer Science*, A. Nerode and Y. V. Matijasevič, eds., *Lecture Notes in Comput. Sci.* 813, Springer, Berlin, 1994, pp. 227–239.
- [23] D. LEIVANT AND J.-Y. MARION, *Lambda calculus characterizations of poly-time*, *Fund. Inform.* 19 (1993), pp. 167–184.
- [24] D. LEIVANT AND J.-Y. MARION, *Ramified recurrence and computational complexity IV: Predicative functionals and poly-space*, *Inform. and Comput.*, to appear.
- [25] A. R. MEYER AND D. M. RITCHIE, *The complexity of loop programs*, in *Proceedings of the 22nd ACM National Conference*, ACM, New York, 1967, pp. 465–469.
- [26] J.-Y. MOYEN, *Analyse de la Complexité et Transformation de Programmes*, Ph.D. thesis, Nancy, 2003. Available online at <http://www-liph.univ-paris13.fr/~moyen/index.html.en>
- [27] MRG (MOBILE RESOURCE GUARANTEES), see <http://groups.inf.ed.ac.uk/mrg/> (2005).
- [28] K.-H. NIGGL, *The μ -measure as a tool for classifying computational complexity*, *Arch. Math. Logic*, 39 (2000), pp. 515–539.
- [29] K.-H. NIGGL, *Control Structures in Programs and Computational Complexity*, Habilitation Thesis, Ilmenau, 2001. Available online at <http://eiche.theoinf.tu-ilmenau.de/~niggl>
- [30] I. OITAVEM, *New recursive characterizations of the elementary functions and the functions computable in polynomial space*, *Rev. Math. Univ. Complut. Madrid*, 10 (1997), pp. 109–125.
- [31] I. OITAVEM, *A term rewriting characterization of the functions computable in polynomial space*, *Arch. Math. Logic*, 41 (2002), pp. 35–47.
- [32] R. W. RITCHIE, *Classes of predictably computable functions*, *Trans. Amer. Math. Soc.*, 106 (1963), pp. 139–173.
- [33] H. SIMMONS, *The realm of primitive recursion*, *Arch. Math. Logic*, 27 (1988), pp. 177–188.

FAST CONSTRUCTION OF NETS IN LOW-DIMENSIONAL METRICS AND THEIR APPLICATIONS*

SARIEL HAR-PELED[†] AND MANOR MENDEL[‡]

Abstract. We present a near linear time algorithm for constructing hierarchical nets in finite metric spaces with constant doubling dimension. This data-structure is then applied to obtain improved algorithms for the following problems: approximate nearest neighbor search, well-separated pair decomposition, spanner construction, compact representation scheme, doubling measure, and computation of the (approximate) Lipschitz constant of a function. In all cases, the running (pre-processing) time is near linear and the space being used is linear.

Key words. doubling dimension, nearest neighbor search, approximate distance oracle, quadtree, metric nets, doubling measure, distance labeling

AMS subject classifications. 68P10, 68W25, 68W40, 52C17

DOI. 10.1137/S0097539704446281

1. Introduction. Given a data set, one frequently wants to manipulate it and quickly compute some properties of it. For example, one would like to cluster the data into similar clusters, or measure similarity of items in the data, etc. One possible way to do this is to define a distance function (i.e., metric) on the data items and perform the required task using this metric. Unfortunately, in general, the metric might be intrinsically complicated (“high-dimensional”), and various computational tasks on the data might require high time and space complexity. This is known in the literature as “the curse of dimensionality.”

One approach receiving considerable attention recently is that of defining a notion of dimension on a finite metric space and developing efficient algorithms for this case. An example of this approach is the notion of doubling dimension [2, 27, 23]. The *doubling constant* of metric space \mathcal{M} is the maximum, over all balls \mathbf{b} in the metric space \mathcal{M} , of the minimum number of balls needed to cover \mathbf{b} , using balls with half the radius of \mathbf{b} . The logarithm of the doubling constant is the *doubling dimension* of the space. The doubling dimension can be thought of as a generalization of the Euclidean dimension, as \mathbb{R}^d has $\Theta(d)$ doubling dimension. Furthermore, the doubling dimension extends the notion of growth restricted metrics of Karger and Ruhl [30].

Understanding the structure of such spaces (or similar notions) and how to manipulate them efficiently received considerable attention in the last few years [14, 30, 23, 28, 34, 33, 44].

The low doubling metric approach can be justified at the following two levels:

1. Arguably, non-Euclidean, low (doubling) dimensional metric data appear in practice and deserve an efficient algorithmic treatment. Even high-dimensional Euclidean data may have some low doubling dimension structure, which makes it amenable to this approach.

*Received by the editors October 11, 2004; accepted for publication (in revised form) October 6, 2005; published electronically March 3, 2006.

<http://www.siam.org/journals/sicomp/35-5/44628.html>

[†]Department of Computer Science, University of Illinois, 201 N. Goodwin Ave., Urbana, IL 61801 (sariel@uiuc.edu, <http://www.uiuc.edu/~sariel>). The work of this author was partially supported by NSF CAREER award CCR-0132901.

[‡]Department of Computer Science, University of Illinois, 201 N. Goodwin Ave., Urbana, IL 61801. Current affiliation: The Open University of Israel, Raanana, Israel (mendelma@gmail.com).

This view seems to be shared by many recent algorithmic papers on doubling metrics, but it still awaits convincing empirical and/or theoretical support.

2. Even if one is only interested in questions on Euclidean point sets, it makes sense to strip the techniques being used to their bare essentials, obtaining a better understanding of the problems and conceptually simpler solutions.

More arguments along these lines can be found in [14], where the author advocates this approach.

In general, it is impossible to directly apply algorithmic results developed for fixed dimensional Euclidean space to doubling metrics, since there exist doubling metrics that cannot be embedded in Hilbert space with low distortion of the distances [41, 35]. Hence, some of the aforementioned works apply notions and techniques from fixed dimensional computational geometry and extend them to finite metric spaces.

In particular, Talwar [44] showed that one can extend the notion of *well-separated pairs decomposition* (WSPD) of [11] to spaces with low doubling dimension. Specifically, he shows that for every set P of n points having doubling dimension \dim , and every $\varepsilon > 0$, there exists WSPD, with separation $1/\varepsilon$ and $O(n\varepsilon^{-O(\dim)} \log \Phi)$ pairs, where \dim is the doubling dimension of the finite metric space, and Φ is the *spread* of the point set, which is the ratio between the diameter of P and the distance between the closest pair of points in P . This is weaker than the result of Callahan and Kosaraju [11] for Euclidean space, which does not depend on the spread of the point set.

Krauthgamer and Lee [34] showed a data structure for answering $(1 + \varepsilon)$ -approximate nearest neighbor queries on point set P with spread Φ . Their data structure supports insertions in $O(\log \Phi \log \log \Phi)$ time. The preprocessing time is $O(n \log \Phi \log \log \Phi)$ (this is by inserting the points one by one), and the query time is $O(\log \Phi + \varepsilon^{-O(\dim)})$. In \mathbb{R}^d for fixed d , one can answer such queries in $O(\log \log(\Phi/\varepsilon))$ time, using near linear space; see [24] and references therein. (In fact, it is possible to achieve constant query time using slightly larger storage [26].) Note, however, that the latter results strongly use the Euclidean structure. Recently, Krauthgamer and Lee [33] overcame the restriction on the spread, presenting a data-structure with nearly quadratic space, and logarithmic query time.

Underlining all those results is the notion of hierarchical nets. Intuitively, hierarchical nets are sequences of larger and larger subsets of the underlining set P , such that in a given resolution, there is a subset in this sequence that represents well the structure of P in this resolution (a formal definition is given in section 2). Currently, the known algorithms for constructing those nets require running time which is quadratic in n .

An alternative way of constructing those nets is by the clustering algorithm of Gonzalez [20]. The algorithm of Gonzalez computes 2-approximate k -center clustering by repeatedly picking the point furthest away from the current set of centers. Setting $k = n$, this results in a permutation of the points in the metric space. It is easy to verify that, by taking different prefixes of this permutation, one gets hierarchical nets for the metric. However, the running time of Gonzalez's algorithm in this case is still quadratic. Although in fixed dimensional Euclidean space the algorithm of Gonzalez was improved to $O(n \log k)$ time by Feder and Greene [17], and to linear time by Har-Peled [25], those algorithms require specifying k in advance, they do not generate the permutation of the points, and as such they cannot be used in this case.

Our results. In this paper, we present for the aforementioned applications improved algorithms having near linear preprocessing time and linear space. We also remove the dependency on the spread. As such, we (almost) match the known results

in computational geometry for low-dimensional Euclidean spaces.

We assume that the input is given via a black box that can compute the distance between any two points in the metric space in constant time. Since the matrix of all $\binom{n}{2}$ distances has quadratic size, this means that in some sense our algorithms have sublinear running time. This is not entirely surprising since subquadratic time algorithms exist for those problems in fixed dimensional Euclidean space. Thus, our paper can be interpreted as further strengthening the perceived connection between finite spaces of low doubling dimensions and Euclidean space of low dimension. Furthermore, we believe that our algorithms for the well-separated pair decomposition and approximate nearest neighbor are slightly cleaner and simpler than the previous corresponding algorithms for the Euclidean case.

Net-tree. In section 3 we present a $2^{O(\dim)}n \log n$ expected time randomized algorithm for constructing the hierarchical nets data-structure, which we call a net-tree.

Approximate nearest neighbor (ANN). In section 4 we show a new data-structure for the $(1 + \varepsilon)$ -approximate nearest neighbor query. The expected preprocessing time is $2^{O(\dim)}n \log n$, the space used is $2^{O(\dim)}n$, and the query time is $2^{O(\dim)} \log n + \varepsilon^{-O(\dim)}$.

This query time is almost optimal in the oracle model since there are examples of point sets in which the query time is $2^{\Omega(\dim)} \log n$ [34], and examples in which the query time is $\varepsilon^{-\Omega(\dim)}$.¹

Our result also matches the results of Arya et al. [1] in Euclidean settings. Furthermore, our result improves upon the recent work of Krauthgamer and Lee, which either assumes bounded spread [34] or requires quadratic space [33]. The algorithms in [1, 34, 33] are deterministic, in contrast to ours.

WSPD. In section 5 we show that one can construct an ε^{-1} WSPD of P in near linear time. The number of pairs is $n\varepsilon^{-O(\dim)}$. The size of the WSPD is tight, as there are examples of metrics in which the size of the WSPD is $n\varepsilon^{-\Omega(\dim)}$. Our result improves upon Talwar’s work [44] and matches the results of Callahan and Kosaraju (the algorithms of both [11, 44] are deterministic, though).

Spanners. A t -spanner of a metric is a sparse weighted graph whose vertices are the metric’s points and in which the graph metric is a t -approximation to the original metric. Spanners were first defined and studied in [40]. Construction of $(1 + \varepsilon)$ -spanners for points in low-dimensional Euclidean space is considered in [31, 10]. Using Callahan’s technique [10], the WSPD construction also implies a near linear time construction of $(1 + \varepsilon)$ -spanners having linear number of edges for such metrics. Independently of our work, Chan et al. [12] show a construction of a $(1 + \varepsilon)$ -spanner for doubling metrics with a linear number of edges. Their construction is stronger in the sense that the degrees in their spanner graph are bounded by constant. However, they do not specify a bound on the running time of their construction.

Compact representation scheme (CRS). In section 6 we construct in near linear time a data-structure of linear size that can answer approximate distance queries between pairs of points in essentially constant time. CRSs were coined “approximate distance oracles” in [45]. Our result extends recent results of Gudmunsson et al. [21, 22], who showed the existence of CRSs with similar parameters for metrics that are “nearly” fixed dimensional Euclidean (and that are a subclass of fixed dou-

¹Consider the set \mathbb{Z}_m^n with the ℓ_∞ norm, where $m = \lceil \varepsilon^{-1}/2 \rceil$, $n = \lceil \dim \rceil$. Consider a query at point q satisfying $\exists x_0 \in \mathbb{Z}_m^n$ such that $d(q, x_0) = m - 1$, and for all $x \in \mathbb{Z}_m^n$, $x \neq x_0 \Rightarrow d(q, x) = m$. Since x_0 can be chosen in an adversarial way, any randomized $(1 + \varepsilon)$ -ANN query algorithm would have to make $\Omega(m^n)$ distance queries before hitting x_0 .

bling dimension metrics). We also mention in passing that our CRS technique can be applied to improve and unify two recent results [44, 43] on distance labeling.

Doubling measure. A doubling measure μ is a measure on the metric space with the property that for every $x \in P$ and $r > 0$, the ratio $\mu(\mathbf{b}(x, 2r))/\mu(\mathbf{b}(x, r))$ is bounded, where $\mathbf{b}(x, r) = \{y : d(x, y) \leq r\}$. Vol'berg and Konyagin [47] proved that for finite metrics (and in fact for complete metrics [36]) the existence of a doubling measure is quantitatively equivalent to the metric being doubling. This measure has found some recent algorithmic applications [43], and we anticipate more applications. Following the proof of Wu [48], we present in section 7 a near linear time algorithm for constructing a doubling measure.

Lipschitz constant of a mapping. In section 8 we study the problem of computing the Lipschitz constant of a mapping $f : P \rightarrow B$. In particular, we show how using WSPD makes it possible to approximate the Lipschitz constant of f in near linear time (in $|P|$) when P has a constant doubling dimension (and B is an arbitrary metric). We also obtain efficient exact algorithms, with near linear running time, for the case where P is a set of points in one- or two-dimensional Euclidean space.

Computing the doubling dimension. Although not stated explicitly in what follows, we assume in sections 2–8 that the doubling dimension of the given metric is either known a priori or given as part of the input. This assumption is removed in section 9, where we remark that a constant approximation of the doubling dimension of a given metric \mathcal{M} can be computed in $2^{O(\dim)} n \log n$ time. It is therefore possible to execute the algorithms of sections 2–8 with the same asymptotic running time, using the approximation of the doubling dimension. (In all the cases where the doubling dimension is needed, any upper bound on it will do, with accordingly degraded running time.)

Most of the algorithms in this paper are randomized. However, our use of randomness is confined to Lemma 2.4 (except for section 8.1). This means that the algorithms always return the desired result, with bounds on the *expected* running time. This also gives the same asymptotic bound with constant probability, using Markov inequality. Furthermore, in the ANN and CRS schema, randomness is used only in the preprocessing, and the query algorithms are deterministic. Lemma 2.4 can be easily derandomized in $O(n^2)$ time, thus giving $n^2 \text{polylog}(n)$ deterministic algorithms for all problems discussed here. We do not know whether a nontrivial derandomization is possible.

2. Preliminaries. Denote by \mathcal{M} a metric space and by P a finite subset $P \subset \mathcal{M}$. The *spread* of P , denoted by $\Phi(P)$, is the ratio of the diameter of P and the distance between the closest pair of points in P . For a point $p \in \mathcal{M}$ and a number $r \geq 0$, we denote by $\mathbf{b}(p, r) = \{q \in \mathcal{M} \mid d_{\mathcal{M}}(p, q) \leq r\}$ the ball of radius r around p . The *doubling constant* λ of P , defined as the minimum over $m \in \mathbb{N}$ such that every ball \mathbf{b} in P , can be covered by at most m balls of at most half the radius. The doubling dimension of the metric space is defined as $\log_2 \lambda$. A slight variation of the doubling constant is that any subset can be covered by λ' subsets of at most half the diameter. It is not hard to see that $\log_2 \lambda$ and $\log_2 \lambda'$ approximate each other up to a factor of 2. Since we will ignore constant factors in the dimension, these two definitions are interchangeable. It is clear that $\log_2 \lambda'(P) \leq \log_2 \lambda'(\mathcal{M})$, and thus the doubling dimension of P is “approximately” at most that of \mathcal{M} .

A basic fact about the λ doubling metric \mathcal{M} that will be used repeatedly is that if $P \subset \mathcal{M}$ has spread at most Φ , then $|P| \leq \lambda^{O(\log_2 \Phi)}$.

2.1. Hierarchy of nets. An r -net in a metric space \mathcal{M} is a subset $\mathcal{N} \subset \mathcal{M}$ of points such that $\sup_{x \in \mathcal{M}} d_{\mathcal{M}}(x, \mathcal{N}) \leq r$ and $\inf_{x, y \in \mathcal{N}; x \neq y} d_{\mathcal{M}}(x, y) \geq r/\alpha$ for some constant $\alpha \geq 1$. r -nets are useful “sparse” object that approximately capture the geometry of the metric space at scales larger than $3r$. In this paper we will rely heavily on the following notion of hierarchical nets.

DEFINITION 2.1 (net-tree). Let $P \subset \mathcal{M}$ be a finite subset. A net-tree of P is a tree T whose set of leaves is P . We denote by $P_v \subset P$ the set of leaves in the subtree rooted at a vertex $v \in T$. Associate with each vertex v a point $\text{rep}_v \in P_v$. Internal vertices have at least two children. Each vertex v has a level $\ell(v) \in \mathbb{Z} \cup \{-\infty\}$. The levels satisfy $\ell(v) < \ell(\bar{p}(v))$, where $\bar{p}(v)$ is the parent of v in T . The levels of the leaves are $-\infty$. Let τ be some large enough constant, say $\tau = 11$.

We require the following properties from T :

Covering property: For every vertex $v \in T$,

$$\mathbf{b}\left(\text{rep}_v, \frac{2\tau}{\tau-1} \cdot \tau^{\ell(v)}\right) \supset P_v.$$

Packing property: For every nonroot vertex $v \in T$,

$$\mathbf{b}\left(\text{rep}_v, \frac{\tau-5}{2(\tau-1)} \cdot \tau^{\ell(\bar{p}(v))-1}\right) \cap P \subset P_v.$$

Inheritance property: For every nonleaf vertex $u \in T$, there exists a child $v \in T$ of u such that $\text{rep}_u = \text{rep}_v$.

The net-tree can be thought of as a representation of nets from all scales in the following sense.

PROPOSITION 2.2. Given a net-tree, let

$$\mathcal{N}_C(l) = \{\text{rep}_u \mid \ell(u) < l \leq \ell(\bar{p}(u))\}.$$

Then the points in $\mathcal{N}_C(l)$ are pairwise $\tau^{l-1}/4$ separated; that is, for any $p, q \in \mathcal{N}_C(l)$, we have $d_{\mathcal{M}}(p, q) \geq \tau^{l-1}/4$. In addition, $P \subseteq \cup_{p \in \mathcal{N}_C(l)} \mathbf{b}(p, 4 \cdot \tau^l)$.

Proof. Let $p, q \in \mathcal{N}_C(l)$, and let u and v be the corresponding nodes in the net-tree, respectively. Consider the balls $\mathbf{b}_p = \mathbf{b}(p, r_p)$ and $\mathbf{b}_q = \mathbf{b}(q, r_q)$, where $r_p = \frac{\tau-5}{2(\tau-1)} \cdot \tau^{\ell(\bar{p}(u))-1}$ and $r_q = \frac{\tau-5}{2(\tau-1)} \cdot \tau^{\ell(\bar{p}(v))-1}$. The sets $\mathbf{b}_p \cap P$ and $\mathbf{b}_q \cap P$ are fully contained in P_u and P_v , respectively, by the definition of the net-tree. Since u and v are on different branches of the net-tree, P_u and P_v are disjoint. But then $d_{\mathcal{M}}(p, q) \geq \max\{r_p, r_q\} \geq \frac{\tau-5}{2(\tau-1)} \cdot \tau^{l-1} \geq \tau^{l-1}/4$ by the definition of $\mathcal{N}_C(l)$ and since $\tau = 11$.

Similarly, consider the set of nodes $V_C(l) = \{u \mid \ell(u) < l \leq \ell(\bar{p}(u))\}$ realizing $\mathcal{N}_C(l)$. For any $v \in V_C(l)$, we have $P_v \subseteq \mathbf{b}(\text{rep}_v, \frac{2\tau}{\tau-1} \cdot \tau^{\ell(v)}) \subseteq \mathbf{b}(\text{rep}_v, \frac{2\tau^l}{\tau-1}) \subseteq \mathbf{b}(\text{rep}_v, \tau^l)$ since $\tau \geq 3$. Thus, $P \subseteq \cup_{v \in V_C(l)} \mathbf{b}(\text{rep}_v, \tau^l) = \cup_{p \in \mathcal{N}_C(l)} \mathbf{b}(p, \tau^l)$, as required. \square

Although $\mathcal{N}_C(\cdot)$ are quantitatively weaker nets compared with the greedy approach,² they are stronger in the sense that the packing and the covering properties respect the hierarchical structure of the net-tree.

The packing and covering properties easily imply that each vertex has at most $\lambda^{O(1)}$ children. Net-trees are roughly equivalent to compressed quadtrees [1]. The

²We have made no attempt to optimize the ratio between the packing and covering radii, and the one reported here can be (substantially) improved. However, some degradation in this ratio seems to be unavoidable.

net-tree is also similar to the sb data-structure of Clarkson [15], but our analysis and guaranteed performance are new.

2.2. The computational model. The model of computation we use is the “unit cost floating-point word RAM model.” More precisely, for a given input consisting of $\text{poly}(n)$ real numbers at the range $[-\Phi, -\Phi^{-1}] \cup [\Phi^{-1}, \Phi]$, and given an accuracy parameter $t \in \mathbb{N}$, the RAM machine has words of length $O(\log n + \log \log \Phi + t)$. These words can accommodate floating-point numbers from the set

$$\left\{ \pm(1+x)2^y \mid x \in [0, 1], x2^{-t} \in \mathbb{N}, y \in [-n^{O(1)} \log^{O(1)} \Phi, n^{O(1)} \log^{O(1)} \Phi] \cap \mathbb{Z} \right\}$$

and integers from the set $\{-(2^t n \log \Phi)^{O(1)}, \dots, 0, \dots, (2^t n \log \Phi)^{O(1)}\}$. For simplicity, we assume that the input given in this way is *exact*. All the problems discussed in this paper have an accuracy parameter of $\varepsilon > 0$. We assume that $\varepsilon^{O(1)} > 2^{-t}$ to avoid rounding problems. The space used by an algorithm (or a scheme) is the number of *words* being used. The machine allows arithmetic, floor, ceiling, conversion from integer to floating point, logarithm, and exponent operations in unit time. We further assume that the machine is equipped with a random number generator.

Floating-point computation is a very well studied topic; see [32, Chap. 4] and references therein. However, we were unable to trace a citation that explicitly defines an asymptotic floating-point computational model. We choose this model for the following two related reasons:

1. The algorithms in this paper are supposed to output only an approximate solution. Therefore it makes sense to try to use approximate numbers since they use less resources.
2. An important theme in this paper is developing algorithms that are independent of the spread of the given metrics. Most algorithms that have an explicit dependence on the spread in their time or space complexity have some form of $\text{polylog}(\Phi)$ dependence. An algorithm that has no dependence on the spread Φ , but relies on words of length $O(\log \Phi)$, may be considered suspicious at best.

Having stated these reasons, for the most part in what follows we will ignore numerical and accuracy issues in our algorithms. The algorithms are simple enough that it is evidently clear that no numerical stability issues arise. A notable exception is Assouad’s embedding discussed in section 6.2. There we have to explicitly add another ingredient (Lemma 6.7) to the algorithm in order to adapt it to the floating-point word RAM model. Indeed, that section is the catalyst for the current discussion.

2.3. Finding a separating ring. We next present a simple argument that helps to overcome the dependence on the spread in the running time.

PROPOSITION 2.3. *Denote by $r_{\text{opt}}(P, m)$ the radius of the smallest ball in P (whose center is also in P) containing m points. Then in a metric space with doubling constant λ , any ball of radius $2r$, where $r \leq 2r_{\text{opt}}(P, m)$, contains at most $\lambda^2 m$ points.*

Proof. By the doubling property, the ball of radius $2r$ can be covered by λ^2 balls of radius $r_{\text{opt}}(P, m)$. Each such ball contains at most m points. \square

LEMMA 2.4. *Given an n -point metric space P with doubling constant λ , one can compute a ball $\mathbf{b} = \mathbf{b}(p, r)$, such that \mathbf{b} contains at least $m = n/(2\lambda^3)$ points of P , and $\mathbf{b}(p, 2r)$ contains at most $n/2$ points of P . The expected running time of this algorithm is $O(\lambda^3 n)$.*

Proof. Pick randomly a point p from P , and compute the ball $\mathbf{b}(p, r)$ of smallest radius around p containing at least $n/(2\lambda^3)$ points. Next, consider the ball of radius

$\mathbf{b}(p, 2r)$. If it contains $\leq n/2$ points, we are done. Otherwise, we repeat this procedure until we succeed.

To see why this algorithm succeeds with constant probability in each iteration, consider the smallest ball $Q = P \cap \mathbf{b}(q, r_{\text{opt}})$ that contains at least m points of P . Observe that any ball of radius $r_{\text{opt}}/2$ contain less than m points. With probability $1/(2\lambda^3)$ our sample is from Q . If $p \in Q$, then $r \leq 2r_{\text{opt}}$, and by the doubling property the ball $\mathbf{b}(p, 4r_{\text{opt}})$ can be covered by at most λ^3 balls of radius $r_{\text{opt}}/2$. Hence it holds that $|P \cap \mathbf{b}(p, 2r)| < \lambda^3 m \leq n/2$.

Thus, the algorithm succeeds with probability $1/(2\lambda^3)$ in each iteration, and with probability $\geq 1/3$ after $2\lambda^3$ iterations, implying the result, as each iteration takes $O(n)$ time. \square

Lemma 2.4 enables us to find a sparse ring of radius “not much larger” than its width. For example, by using it we can find an empty ring of width h and radius at most $2nh$ in linear time.

3. Computing nets efficiently.

In this section we prove the following theorem.
THEOREM 3.1. *Given a set P of n points in \mathcal{M} , one can construct a net-tree for P in $2^{O(\dim)} n \log n$ expected time.*

The outline of the proof is as follows. In section 3.1 we show how to construct the Gonzalez sequence in $2^{O(\dim)} n \log(n + \Phi)$ time. We then eliminate the dependence of the running time on the spread Φ in section 3.3 by using a tool developed in section 3.2. In section 3.4 we conclude the proof of Theorem 3.1 by showing how to construct the net-tree from the Gonzalez sequence. We end by mentioning in section 3.5 a few data structures for efficient searching on the net-tree.

3.1. Computing greedy clustering quickly. Gonzalez [20] presented a greedy algorithm, denoted by **GreedyCluster**, that when applied to a set of points P computes a permutation of the points $\Pi = \langle p_1, p_2, \dots, p_m \rangle$, such that p_1, \dots, p_k are good centers for P , for any $k \geq 1$. We refer to Π as the *greedy permutation* of P . Formally, there are numbers r_1, \dots, r_n , such that $P \subseteq \cup_{i=1}^k \mathbf{b}(p_i, r_k)$. Furthermore, $\min_{1 \leq i < j \leq k} d_{\mathcal{M}}(p_i, p_j) = r_{k-1}$.

GreedyCluster works by picking an arbitrary point in P to be p_1 and setting r_1 to be the distance of the furthest point in P to p_1 . For every point $q \in P$, **GreedyCluster** stores its distance to the closest center picked so far; namely, in the beginning of the k th iteration, for all $q \in P$ we have $\alpha_q^k = \min_{i=1}^{k-1} d_{\mathcal{M}}(q, p_i)$. The algorithm sets the k th center to be $p_k = \arg \max_{p \in P} \alpha_p^k$ (namely, p_k is the point in P furthest away from the centers picked so far). Clearly, $r_{k-1} = \alpha_{p_k}^k$. By implementing this naively, one can compute the first k points p_1, \dots, p_k in $O(nk)$ time. Thus, this leads to a 2-approximation to k -center clustering in $O(nk)$ time.

Feder and Greene [17] improved the running time to $O(n \log k)$ time (this was further improved to linear time by Har-Peled [25]). Feder and Greene’s main observation was that when updating α_q^{k+1} , one needs to update this value only for points of P , which are in distance $\leq r_{k-1}$ away from p_k , since for points q further away, the addition of p_k cannot change α_q^k .

This suggests the following natural approach for computing the greedy permutation: Associate with each center in $\{p_1, \dots, p_k\}$ the points of P that it serves (namely, points that are closer to the given center than to any other center). Furthermore, each center p_i maintains a *friends list* that contains all the centers that are a distance of at most $4r_k$ from it. An “old” center will trim a point from its friends list only when its distance is larger than $8r_k$. Specifically, the friends list of p_i at the k th iteration

($k \geq i$) contains all the centers at a distance of at most $\min\{8r_k, 4r_i\}$ from p_i . Because of the constant doubling dimension property, this list is of size $\lambda^{O(1)}$.

We further maintain a max-heap in which every center p_i , $i < k$, maintains the point p'_i furthest away from p_i in its cluster along with its current $\alpha_{p'_i} = d_{\mathcal{M}}(p_i, p'_i)$ value.

At the k th iteration, the algorithm extracts the maximum value from the heap. It sets p_k to be the corresponding point. Denote by c_{p_k} the closest point among $\{p_1, \dots, p_{k-1}\}$ to p_k (i.e., the cluster's center of p_k at the end of the $(k-1)$ th round). Next, the algorithm scans all the points currently served by the same cluster as c_{p_k} , or by clusters containing points from friends list of c_{p_k} , and updates the α value of those points. Furthermore, it moves all the relevant points to the newly formed cluster. In the process, it also update the points p'_i (of maximum distance from p_i in its cluster) for all p_i in the friends list of c_{p_k} . It also computes the friends list of p_k (how exactly this is done will be described in detail shortly).

We next bound the running time. To this end, a phase starting at the i th iteration of the algorithm terminates at the first $j > i$ such that $r_{j-1} \leq r_{i-1}/2$. A ball of radius $4r_{j-1}$ around each point $q \in P$ contains at most λ^3 points of p_1, \dots, p_j , and as such every point of P is being scanned at most λ^3 times at each phase of the algorithm. Thus, if the spread of the point set is Φ , the number of phases is $O(\log \Phi)$, and scanning takes $\lambda^{O(1)}n \log \Phi$ time overall. Maintaining the max-heap costs an additional $\lambda^{O(1)}n \log n$ time, since in each iteration only $\lambda^{O(1)}$ values in the head are changed.

The only remaining hurdle is the computation of the friends list of a newly formed center p_k . This can be done by maintaining, for every point p_l , $l \in \{1, \dots, n\}$, the serving center $p_{l'}$ two phases ago (at the end of that phase). The friends list of p_k is constructed by scanning the friends list of $p_{k'}$ and picking those points that are at distance at most $4r_k$ from p_k . This costs $\lambda^{O(1)}$ time for p_k and $O(\lambda^{O(1)}n)$ time overall. To see that this search suffices, note that the set $\{p_i | i < k, d_{\mathcal{M}}(p_i, p_k) \leq 4r_k\}$ is scanned. Indeed, fix p_{i_0} , having $i_0 < k$, and $d_{\mathcal{M}}(p_{i_0}, p_k) \leq 4r_k$. Let $p_{k'}$ be the center of p_k two phases ago. From the definition, $2r_k \leq r_{k'} \leq 4r_k$, and thus $d_{\mathcal{M}}(p_k, p_{k'}) \leq 4r_k$. The current (at the end of the $(k-1)$ th iteration) friends list of $p_{k'}$ contains all the current centers at a distance of at most $\min\{8r_k, 4r_{k'}\} = 8r_k$ from $p_{k'}$. Furthermore,

$$d_{\mathcal{M}}(p_{i_0}, p_{k'}) \leq d_{\mathcal{M}}(p_{i_0}, p_k) + d_{\mathcal{M}}(p_k, p_{k'}) \leq 8r_k.$$

We are therefore guaranteed that p_{i_0} will be scanned.

Of course, as the algorithm progresses it needs to remove nonrelevant elements from the friends list as the current clustering radius r_i shrinks. However, this can be done in a lazy fashion whenever the algorithm scans such a list.

THEOREM 3.2. *Let P be an n -point metric space with doubling constant λ and spread Φ . Then the greedy permutation for P can be computed in $O(\lambda^{O(1)}n \log(\Phi n))$ time and $O(\lambda^{O(1)}n)$ space.*

3.2. Low quality approximation by HST. Here we present an auxiliary tool that will be used in section 3.3 to extend the net-tree construction of section 3.1 to metric spaces with large spread.

We will use the following special type of metric spaces.

DEFINITION 3.3. *A hierarchically well-separated tree (HST) is a metric space defined on the leaves of a rooted tree T . Associated with each vertex $u \in T$ is a label $\Delta_u \geq 0$ such that $\Delta_u = 0$ if and only if u is a leaf of T . The labels are such that if a vertex u is a child of a vertex v , then $\Delta_u \leq \Delta_v$. The distance between two leaves x*

and y of T is defined as $\Delta_{lca(x,y)}$, where $lca(x,y)$ is the least common ancestor of x and y in T .

The class of HSTs coincides with the class of finite ultrametrics. For convenience, we will assume that the underlying tree is binary (any HST can be converted into a binary HST in linear time, while retaining the underlying metric). We will also associate with every vertex $u \in T$ an arbitrary leaf rep_u of the subtree rooted at u . We also require that $\text{rep}_u \in \{\text{rep}_v \mid v \text{ is a child of } u\}$.

A metric N is called a t -approximation of the metric \mathcal{M} if N and \mathcal{M} are defined on the same set of points and $\forall u, v \in \mathcal{M}, d_{\mathcal{M}}(u, v) \leq d_N(u, v) \leq t \cdot d_{\mathcal{M}}(u, v)$.

It is not hard to see that any n -point metric is $(n - 1)$ -approximated by some HST (see, e.g., Lemma 3.6). Here we show the following.

LEMMA 3.4. *For an n -point metric space \mathcal{M} with doubling constant λ , it is possible to construct in $O(\lambda^6 n \log n)$ expected time an HST which is a $3n^2$ approximation of \mathcal{M} .*

This low quality HST will help us later in eliminating the dependence on the spread of the construction time of the net-tree and in distance queries.

We begin proving Lemma 3.4 by constructing a sparse graph that approximates the original metric (this is sometimes called a *spanner*).

LEMMA 3.5. *Given an n -point metric space P with doubling constant λ , one can compute a weighted graph G that $3n$ -approximates P in $O(\lambda^6 n \log n)$ expected time. The graph G contains $O(\lambda^3 n \log n)$ edges.*

Proof. The construction is recursive. If $n = O(1)$, we just add all the pairs from P as edges. Otherwise, we compute, using Lemma 2.4, a ball $\mathbf{b}(c, r)$ containing at least $m = n/(2\lambda^3)$ points of P with the additional property that $\mathbf{b}(c, 2r)$ contains at most $n/2$ points of P .

As such, there exists two numbers r', h such that $r \leq r' \leq 2r$, $h \geq r/n$, and $P \cap \mathbf{b}(c, r') = P \cap \mathbf{b}(c, r' + h)$ (namely, the ring with outer radius $r' + h$ and inner radius r' around c is empty of points of P). Computing r' and h is done by computing the distance of each point from c and partitioning the distance range $[r, 2r]$ into $2n$ segments of equal length. In each segment, we register the point with minimum and maximum distance from c in this range. This can be easily done in $O(n)$ time using the floor function. Next, scan those buckets from left to right. Clearly, the maximum length gap is realized by a maximum of one bucket together with a consecutive nonempty minimum of another bucket. Thus, the maximum length interval can be computed in linear time, and it yields r and h .

Let $P_{\text{in}} = \mathbf{b}(c, r') \cap P$ and let $P_{\text{out}} = P \setminus P_{\text{in}}$. Observe that $d_{\mathcal{M}}(P_{\text{in}}, P_{\text{out}}) = \min_{p \in P_{\text{in}}, q \in P_{\text{out}}} d_{\mathcal{M}}(p, q) \geq h \geq r/n$. Next, we build recursively a spanner for P_{in} and a spanner for P_{out} . We then add the edges between c and all the points of P to the spanner. Let G denote the resulting graph.

Since there are $n/2 \geq |P_{\text{in}}| \geq n/2\lambda^3$ points of P , the running time of the algorithm is $T(|P|) = T(|P_{\text{in}}|) + T(|P_{\text{out}}|) + O(\lambda^3 n) = O(\lambda^6 n \log n)$. Similarly, the number of edges in G is $O(\lambda^3 n \log n)$.

Remaining is the task of proving that G provides a $3n$ -approximation to the distances of P . Let G_{in} and G_{out} be the graphs computed for P_{in} and P_{out} , respectively. Consider any two points $u, v \in P$. If u and v are both in P_{in} or both in P_{out} , then the claim follows by induction. Thus, consider the case that $u \in P_{\text{in}}$ and $v \in P_{\text{out}}$. Observe that $d_{\mathcal{M}}(u, v) \geq h \geq r/n$. On the other hand,

$$\begin{aligned} r/n \leq d_{\mathcal{M}}(u, v) &\leq d_G(u, v) \leq d_{\mathcal{M}}(c, u) + d_{\mathcal{M}}(c, v) \\ &\leq r + r + d_{\mathcal{M}}(u, v) \leq (2n + 1)d_{\mathcal{M}}(u, v), \end{aligned}$$

since $d_{\mathcal{M}}(c, v) \leq d_{\mathcal{M}}(c, u) + d_{\mathcal{M}}(u, v) \leq r + d_{\mathcal{M}}(u, v)$. Clearly, this implies that $d_G(u, v) \leq 3nd_{\mathcal{M}}(u, v)$, as claimed. \square

We will later obtain in Theorem 5.3 a near linear time construction of spanners that $(1 + \varepsilon)$ -approximate the original metric and have a linear number of edges.

LEMMA 3.6. *Given a weighted connected graph G on n vertices and m edges, it is possible to construct in $O(n \log n + m)$ time an HST H that $(n - 1)$ -approximates the shortest path metric on G .*

Proof. Compute the minimum spanning tree of G in $O(n \log n + m)$ time, and let T denote this tree.

Sort the edges of T in nondecreasing order, and add them to the graph one by one. The HST is built from the bottom up. At each point we have a collection of HSTs, each of which corresponds to a connected component of the current graph. When an added edge merges two connected components, we merge the two corresponding HSTs into one by adding a new common root for the two HSTs and labeling this root with the edge’s weight times $n - 1$. This algorithm is only a slight variation on the Kruskal algorithm and has the same running time.

We next estimate the approximation factor. Let x and y be two vertices of G . Denote by e the first edge that was added in the process above that made x and y to be in the same connected component C . Note that at that point in time, e is the heaviest edge in C , so $w(e) \leq d_G(x, y) \leq (|C| - 1)w(e) \leq (n - 1)w(e)$. Since $d_H(x, y) = (n - 1)w(e)$, we are done. \square

The proof of Lemma 3.4 now follows by applying Lemma 3.6 on the spanner from Lemma 3.5.

Note that by applying Lemma 3.6 on the spanner from Theorem 5.3, one can obtain a near linear time construction of an HST which $O(n)$ -approximates that original metric.

3.3. Extending greedy clustering to metrics of large spread. The main idea in removing the dependence of the running time on the spread is to apply the algorithm of section 3.1 to a *dynamic* set of points that will correspond to a level of the HST. In more detail, the set of points will correspond to the representatives rep_v , where $\Delta_v \leq r_{\text{curr}}/n^4 \leq \Delta_{\bar{p}(v)}$, where r_{curr} is the current greedy radius, Δ_v is the HST label of v (i.e., the diameter of the subtree rooted at v), and $\bar{p}(v)$ is the parent of v in the HST. The algorithm now needs to handle another type of event since, as the algorithm proceeds, the greedy radius decreases to a level in which $\Delta_v \geq r_{\text{curr}}/n^4$. In this case, v should be replaced with its two children u, w . Specifically, if v belongs to a cluster of a point p_i , we remove rep_v from the list of points associated with the cluster of p_i and add rep_u and rep_w to this list (the case where p_i is equal to rep_v is handled in a similar fashion). Next, we need to compute for the new point its nearest center; namely, compute α_{rep_u} and α_{rep_w} (in fact, since $\text{rep}_v = \text{rep}_u$ or $\text{rep}_v = \text{rep}_w$, we need to compute only one of those values). To this end, we scan the friends list of p_i and compute α_{rep_u} and α_{rep_w} from it. This takes $\lambda^{O(1)}$ time. We also need to insert $\{\text{rep}_u, \text{rep}_w\} \setminus \{\text{rep}_v\}$ into the max-heap.

Thus, the algorithm has two heaps. One is a max-heap maintaining the points according to their distances to the nearest center; that is, for every point $p \in P$ we maintain the values of α_p in a max-heap. The second max-heap maintains the nodes of the HST sorted by their diameters Δ (multiplied by a factor of n^4 for normalization). At every point, the algorithm extracts the larger of two heaps and handles it accordingly. One important technicality is that the algorithm is no longer generating the same permutation as **GreedyCluster**, since we are not always picking the furthest

point to add as the next center. Rather, we add the furthest active point. We refer to the new algorithm as **NetPermutAlg**.

LEMMA 3.7. *Let $\pi = \langle p_1, \dots, p_n \rangle$ be the permutation of P generated by **NetPermutAlg**. Furthermore, let $r_k = \alpha_{p_{k+1}}^{k+1} = \min_{i=1}^k d_{\mathcal{M}}(q, p_i)$. Then, $P \subseteq \cup_{i=1}^k \mathbf{b}(p_i, (1 + n^{-2})r_k)$ and for any $u, v \in \{p_1, \dots, p_k\}$ we have $d_{\mathcal{M}}(u, v) \geq (1 - n^{-2})r_k$.*

Proof. Clearly, the balls of radius r_k around p_1, \dots, p_k cover all the active points when p_{k+1} was created. However, every active point might represent points which are a distance of r_k/n^2 from it. Thus, by expanding the radius by $(1 + 1/n^2)$, those balls cover all the points.

Observe, that this implies that for any $i < j$ we have $(1 + n^{-2})r_i \geq r_j$. In particular, let $\alpha \leq k$ be the minimum number such that $u, v \in \{p_1, \dots, p_\alpha\}$. Clearly, $d_{\mathcal{M}}(u, v) \geq r_{\alpha-1} \geq r_\alpha/(1 + n^{-2}) \geq (1 - n^{-2})r_\alpha$. \square

LEMMA 3.8. *The expected running time of **NetPermutAlg** is $O(\lambda^{O(1)}n \log n)$.*

Proof. Constructing the HST takes $\lambda^{O(1)}n \log n$ expected time, using Lemma 3.4. As in the bounded spread case, we conceptually divide the execution of the algorithm into phases. In the i th phase, the algorithm handles new clusters with radii in the ranges $\text{diam}(P)/2^{i-1}$ and $\text{diam}(P)/2^i$. Consider a point $p \in P$: It is inserted into the point-set when a node v in the HST is “split” at phase i (since p is the representative point for one of the children of v). Let p and q be the two representative points of the two children of v . We charge v for any work done with p and q for the next $L = 10 \log n$ phases. Consider any work done on p before it undergoes another split event. If p is at most L phases away from the split event of v , the vertex v pays for it.

Otherwise, consider p at $> L$ phases away from its latest split event that happened at v . Let r_{curr} be the current clustering radius, and observe that p represents a set of points which has a diameter $\leq r_{\text{curr}}/n^2$ and that $r_{\text{curr}} \leq \Delta_v/n^{10}$. In particular, this implies that $P \cap \mathbf{b}(p, r_{\text{curr}} \cdot n^2) \subset P \cap \mathbf{b}(p, \Delta_v/n^4) \subset P \cap \mathbf{b}(p, r_{\text{curr}}/n^2)$. Namely, all the points that p represents are very far from the rest of the points of P , in terms of r_{curr} . In particular, it cannot be that the cluster that p represents is in any updated friends list in the current stage. (It can be in a friends list that was not updated lately, since we use lazy evaluation. However, when this friends list is used, it will be updated and p will disappear from it. Note that the work required to update the friends lists is $\lambda^{O(1)}n$ overall; see section 3.1.) Thus, p does not require any work from the algorithm until it undergoes another split event.

Thus, every node in the HST is charged with $\lambda^{O(1)} \log n$ work. It follows that the overall running time of the algorithm is $\lambda^{O(1)}n \log n$. \square

3.4. Constructing the net-tree. In this section we conclude the description of the algorithm for constructing the net-tree and prove Theorem 3.1.

The construction of the net-tree T is done by adding the points of P according to the permutation of **NetPermutAlg**. As mentioned before, the construction algorithm and the resulting tree are similar to the data-structure of Clarkson [15] (our analysis and the guaranteed performance are new, however). The tree constructed for p_1, \dots, p_k is denoted by $T^{(k)}$ and $T = T^{(n)}$. We obtain $T^{(k)}$ from $T^{(k-1)}$ as follows.

During the construction, we maintain for every vertex $u \in T^{(k)}$ a set of “close by” vertices $\text{Rel}(u)$. Namely, the set $\text{Rel}(u)$ would be in fact the set

$$\overline{\text{Rel}}(u) = \left\{ v \in T^{(k)} \mid \ell(v) \leq \ell(u) < \ell(\overline{p}(v)), \text{ and } d_{\mathcal{M}}(\text{rep}_u, \text{rep}_v) \leq 13 \cdot \tau^{\ell(u)} \right\},$$

where τ is the packing constant associated with the net-tree; see Definition 2.1. (Since we compute $\text{Rel}(u)$ indirectly, the fact that $\text{Rel}(u) = \overline{\text{Rel}}(u)$ requires a formal proof;

see Lemma 3.9(v).) The set $\text{Rel}(u)$ is of size $\lambda^{O(1)}$ throughout the algorithm's execution.

We denote $\bar{r}_i = \min \{r_j \mid 1 \leq j \leq i\}$.

The algorithm. The k th point in the permutation, p_k , will be added as a leaf to the tree $T^{(k-1)}$ to form the tree $T^{(k)}$. As such, we fix $\ell(p_k) = -\infty$ and $\text{rep}_{p_k} = p_k$. Let $l = \lceil \log_\tau \bar{r}_{k-1} \rceil$.

Let h be the largest index such that $\log_\tau \bar{r}_{h-1} > l$ (i.e., p_h is the last added center in the previous phase). Let $q \in \{p_1, \dots, p_h\}$ be the closest point to p_k among $\{p_1, \dots, p_h\}$; namely, q is the nearest neighbor to p_k in all the centers present in the previous phase. Identifying q with the unique leaf of $T^{(k-1)}$ whose representative is q , let $u = \bar{p}(q)$. We obtain $T^{(k)}$ as follows:

- (a) If $\ell(u) > l$, then we make a new vertex v and set $\ell(v) = l$ and $\text{rep}_v = q$. We then connect q and p_k as children of v and make v a child of u .
- (b) Otherwise, connect p_k as another child of u .

Finding q . Let c_{p_k} be the closest point among $\{p_1, \dots, p_{k-1}\}$ to p_k (this information is computed by `NetPermutAlg`; see section 3.1 for details). Denote $\hat{u} = \bar{p}(c_{p_k})$. We consider the following two cases:

- (1) If $\ell(\hat{u}) > l$, then $q = \hat{u}$; see Lemma 3.9(i) for a proof.
- (2) Otherwise, $\ell(\hat{u}) = l$. In this case, q must be in the set $\{\text{rep}_w \mid w \in \text{Rel}(\hat{u})\}$; see Lemma 3.9(i) for a proof. Thus, we just pick q to be the nearest neighbor to p_k in $\{\text{rep}_w \mid w \in \text{Rel}(\hat{u})\}$.

Updating $\text{Rel}(\cdot)$. For each new vertex x added we do the following. Let $y = \bar{p}(x)$. For each $z \in \text{Rel}(y)$, and for each child z' of z , we traverse *part* of the tree rooted at z' in the following way: When visiting a vertex u , we check whether u should be added to $\text{Rel}(x)$ and whether x should be added to $\text{Rel}(u)$ according to the $\overline{\text{Rel}}(\cdot)$ definition, and update $\text{Rel}(x)$ and $\text{Rel}(u)$ accordingly. If x has been added to $\text{Rel}(u)$, then we continue by traversing the children of u . Otherwise, we skip them.

Note that this might require scanning a large fraction of the net-tree, as x might appear in a large number of $\text{Rel}()$ lists.

LEMMA 3.9. *For any $k \in [1, \dots, n]$, the tree $T^{(k)}$ has the following properties:*

- (i) *The part of the algorithm that finds q indeed finds it.*
- (ii) *If v is a child of u , then $d_{\mathcal{M}}(\text{rep}_u, \text{rep}_v) \leq 2 \cdot \tau^{\ell(u)}$.*
- (iii) *For every $t \in \mathfrak{R}$, every pair of points in $\mathcal{N}_C(t)$ is at least τ^{t-1} far apart.*
- (iv) *$T^{(k)}$ is a net-tree of $\{p_1, \dots, p_k\}$.*
- (v) *For any $u \in T$, $\text{Rel}(u) = \overline{\text{Rel}}(u)$.*

Since the proof of Lemma 3.9 is tedious, we defer it to the appendix. We next analyze the running time.

LEMMA 3.10. *Given the (approximate) greedy permutation $\langle p_1, \dots, p_n \rangle$ with its "current" cluster's center $\langle c_{p_2}, \dots, c_{p_n} \rangle$, the algorithm for constructing the net-tree runs in $\lambda^{O(1)}n$ time.*

Proof. By the definition of $\text{Rel}(\cdot)$, the size of each such list is at most $\lambda^{O(1)}$. Assuming the tree is implemented reasonably (with pointers from a vertex to its children and parent), constructing the tree clearly takes $O(\lambda^{O(1)})$ time per new point.

Next we estimate the time to construct $\text{Rel}(\cdot)$. For each vertex x added, we first charge $\lambda^{O(1)}$ visits for visiting the children of $\text{Rel}(\bar{p}(x))$. All the other visits are charged to the parent of the visited vertex. Each vertex has at most $\lambda^{O(1)}$ children, and its children are visited only if a new entry was inserted into its $\text{Rel}()$. As the total size of the $\text{Rel}(\cdot)$ lists is at most $\lambda^{O(1)}n$, we have just bounded the number of visits of vertices during the update process of $\text{Rel}(\cdot)$ to $\lambda^{O(1)}n$. Thus the time spent is $\lambda^{O(1)}n$. \square

3.5. Augmenting the net-tree. In order to efficiently search on the net-tree, we will need the following three auxiliary data structures.

The first one, given a vertex v of level l , allows us to find all the vertices of “roughly the same level” that are nearby, i.e., those with a representative at a distance of at most $O(\tau^l)$ from the representative of v . More accurately, we need fast access to $\overline{\text{Rel}}(v)$, as defined in section 3.4. We have seen in that section how to construct it in near linear time such that the whole list can be accessed in $O(\lambda^4)$ time.

The second data-structure enables the following seek operation: Given a leaf x and a level l , find the ancestor y of x such that $\ell(\overline{p}(y)) > l \geq \ell(y)$. Bender and Farach-Colton [5] present a data-structure \mathcal{D} that can be constructed in linear time over a tree T such that, given a node x and depth d , it outputs the ancestor of x at depth d at x . This takes constant time per query. Thus, performing the seek operation just requires performing a binary search using \mathcal{D} over the net-tree, and this takes $O(\log n)$ time.

Our third data-structure supports a restricted version of the above seek operation: Given a leaf x , an ancestor z of x , and a level l : If $l \notin [\ell(z) - c \log n, \ell(z)]$, return “don’t know.” Otherwise, return an ancestor y of x satisfying $\ell(\overline{p}(y)) > l \geq \ell(y)$ (here $c > 0$ is an absolute constant). The data-structure has $O(n)$ space and $O(n \log n)$ preprocessing time, and the queries can be answered in *constant time*.

As a first step, observe that if for every internal vertex z and a descendant leaf x we add vertices to the tree so as to fill all levels between $\ell(z)$ and $\ell(x) - c \log n$ on the path between z and x , then queries to the l level ancestor, $l \in [\ell(x) - c \log n, \ell(z)]$, can be answered by using the data-structure \mathcal{D} as above to find an ancestor of x at depth $\overline{d}(z) - (\ell(z) - l)$. This construction, however, may blow up the number of vertices in the net-tree (and hence the space) by a $\log n$ factor.

To obtain linear space we do the following: In the preprocessing step we enumerate all the possible patterns of existence/nonexistence of vertices in $0.5 \log_2 n$ consecutive levels. For each given pattern and each given level in the pattern, we write the number of actual vertices above this level. Preparing this enumeration takes only $O(\sqrt{n} \log n)$ time. Now, for each vertex u of the net-tree, we hold $2c$ pointers to such patterns that together map the vertices in the $c \log n$ level below v on the path to u , where v is an ancestor of u at depth $\overline{d}(u) - c \log n$, if such v exists (note that v is $c \log n$ edges above u in the net-tree, but u holds the pattern of only the first $c \log n$ levels below v). This data-structure can be clearly computed in $O(n \log n)$ time using top-down dynamic programming on the net-tree.

Given a query (with x , z , and l as above), we do as follows: Let u be an ancestor of x at depth $\max\{\overline{d}(z) + c \log n, \overline{d}(x)\}$. Vertex u can be accessed in $O(1)$ time using the data-structure \mathcal{D} . Using the patterns pointed out by u , we can find the depth of the relevant vertex whose level is just below l in $O(1)$ time, and now using \mathcal{D} again we can access this vertex in constant time.

4. Approximate nearest neighbor search. In the following, ANN stands for approximate nearest neighbor. In this section, we present an ANN scheme that preprocesses a given set of points P in near linear time and produces a linear space data-structure which answers queries of the form “given point q , find $p \in P$ such that $d(q, p) \leq (1 + \epsilon)d(q, P)$ ” in logarithmic time. See section 1 for more details.

In section 4.1, we present a variant of Krauthgamer and Lee’s [34] net navigation algorithm for the net-tree. This algorithm allows us to boost an A -ANN solution to a $(1 + \epsilon)$ -ANN solution in $O(\log n + \log A + \epsilon^{O(\dim)})$ query time. In section 4.2 we present a fast construction of a variant of the ring separator tree [29, 33] which

supports fast $2n$ -ANN queries. We conclude in section 4.3 with the general scheme which is a combination of the previous two.

4.1. The low spread case.

LEMMA 4.1. *We are given a net-tree T of P , a query point $q \in \mathcal{M}$, and a vertex $u \in T$ at level $l = \ell(u)$ such that $d_{\mathcal{M}}(\text{rep}_u, q) \leq 5 \cdot \tau^l$ or $\widehat{p} \in P_u$, where \widehat{p} is the nearest neighbor to q in P . Then there is an algorithm that traverses T from u downward such that for any $t \in \mathbb{N}$, after $t + 4$ steps, the algorithm reaches a vertex s for which rep_s is a $(1 + \tau^{l-f-t})$ -ANN, where $f = \log_{\tau} d_{\mathcal{M}}(\widehat{p}, q)$. The running time of this search is $\lambda^{O(1)} \min\{t, l - f\} + \lambda^{O(\max\{t-(l-f), 0\})}$.*

Proof. The query algorithm works as follows. It constructs sets A_i of vertices in T with the following properties:

1. For each $v \in A_i$, $\ell(\overline{p}(v)) > i \geq \ell(v)$.
2. $\widehat{p} \in \cup_{v \in A_i} P_v \subset \mathbf{b}(q, d_{\mathcal{M}}(q, \widehat{p}) + (13 + \frac{2\tau}{\tau-1}) \cdot \tau^i)$.

The algorithm starts by setting $A_l = \text{Rel}(u)$. If $\widehat{p} \in P_u$, then A_l clearly satisfies the two properties above. If $d_{\mathcal{M}}(\text{rep}_u, q) \leq 5 \cdot \tau^l$, then $d_{\mathcal{M}}(\text{rep}_u, \widehat{p}) \leq 10\tau^l$. Suppose for the sake of contradiction that $\widehat{p} \notin \cup_{v \in A_l} P_v$; then $\exists v'$ such that $\ell(v') \leq l$, $d_{\mathcal{M}}(\text{rep}_u, \text{rep}_{v'}) > 13\tau^l$, and $\widehat{p} \in P_{v'}$. But then from the covering property, $d_{\mathcal{M}}(\text{rep}_{v'}, \widehat{p}) \leq \frac{2\tau}{\tau-1} \tau^l$, which means that $d_{\mathcal{M}}(\text{rep}_u, \widehat{p}) > (13 - \frac{2\tau}{\tau-1})\tau^l > 10\tau^l$, a contradiction.

The set A_{i-1} is constructed from A_i as follows. Let $v \in A_i$ be the closest vertex in A_i to q , i.e., $d_{\mathcal{M}}(\text{rep}_v, q) = \min_{w \in A_i} d_{\mathcal{M}}(\text{rep}_w, q)$. Let B be the set obtained from A_i by replacing every vertex of level i with its children. The set A_{i-1} is obtained from B by throwing out any vertex w for which $d_{\mathcal{M}}(q, \text{rep}_w) > d_{\mathcal{M}}(q, \text{rep}_v) + \frac{2\tau}{\tau-1} \cdot \tau^{i-1}$. It is easily checked that A_{i-1} has the required properties.

The running time is clearly dominated by $\lambda^{O(1)}$ times the sum of the A_i 's sizes. For $i > f$, $d_{\mathcal{M}}(q, \text{rep}_v)$ is at most $\frac{2\tau}{\tau-2} \cdot \tau^i$, and therefore $|A_i| \leq \lambda^{O(1)}$. For $i \leq f$, we have only a weak bound of $|A_i| \leq \lambda^{O(f-i)}$. Thus the running time of the algorithm for t steps follows. Notice that any point in A_{l-i} is $(1 + \tau^{l-f-i+4})$ -ANN. \square

For a set P with spread Φ , by applying the algorithm of Lemma 4.1 with u the root of T , and $t = \lceil \log_{\tau}(\Phi/\varepsilon) - f \rceil$, Lemma 4.1 gives a $(1 + \varepsilon)$ -approximate nearest neighbor scheme with $O(n \log n)$ expected construction time and $O(\log \Phi + \varepsilon^{-O(\dim)})$ query time. (Note that the algorithm does not need to know t (and thus f) in advance—it can estimate the current approximation by comparing $d_{\mathcal{M}}(q, \text{rep}_v)$ to τ^i .) This gives an alternative to the data-structure of Krauthgamer and Lee [34], with a slightly faster construction time. Their construction time is $O(n \log \Phi \log \log \Phi)$ if one uses the insertion operation for their data-structure (note that in the constant doubling dimension setting, $\log n = O(\log \Phi)$). In fact, in this case, the $\text{Rel}()$ data-structure is not needed since $\text{Rel}(\text{root}) = \{\text{root}\}$. Therefore the storage for this ANN scheme is $O(n)$, with no dependency on the dimension. A similar construction was obtained independently in [8]. However, its construction time is $O(n^2)$.

4.2. Low quality ring separator tree.

LEMMA 4.2. *One can construct a data-structure which supports $2n$ -ANN queries in $2^{O(\dim)} \log n$ time. The construction time is $2^{O(\dim)} n \log n$, and the data-structure uses $2^{O(\dim)} n$ space.*

Proof. The data structure is a binary search tree S , in which each vertex of the tree v is associated with a point $p_v \in P$ and radius r_v . We are guaranteed that $n/2\lambda^3 \leq |\mathbf{b}(p_v, r_v)| \leq (1 - 1/2\lambda^3)n$ and that $(\mathbf{b}(p_v, (1 + 1/2n)r_v) \setminus \mathbf{b}(p_v, (1 - 1/2n)r_v)) \cap P = \emptyset$. The left subtree is recursively constructed on the set $P \cap \mathbf{b}(p_v, r_v)$, and the right

subtree is recursively constructed on $P \setminus \mathbf{b}(p_v, r_v)$. The depth of S is clearly at most $O(\lambda^3 \log n)$.

The construction of S is similar to the construction of the low quality spanner (section 3.2) and uses Lemma 2.4 as follows. Apply Lemma 2.4 to find $p \in P$ and r such that $|\mathbf{b}(p, r)| \geq n/(2\lambda^3)$, whereas $|\mathbf{b}(p, 2r)| \leq n/2$. From the pigeonhole principle, there exists $r' \in [(1 + 1/2n)r, 2r - r/2n)$ for which $\mathbf{b}(p, (1 + 1/2n)r') \setminus \mathbf{b}(p, (1 - 1/2n)r') = \emptyset$. We now make a root v for the ring separator tree, set $p_v = p$ and $r_v = r'$, and recurse on $\mathbf{b}(p_v, r_v)$ as the left subtree and $P \setminus \mathbf{b}(p_v, r_v)$ as the right subtree. The construction time $T(n)$ obeys the recursive formula $T(n) = T(n_1) + T(n_2) + O(n)$, where $n_1 + n_2 = n$, $n/2\lambda^3 \leq n_1 \leq n/2$.

Once we have this data-structure, $2n$ -ANN can be found in $O(\lambda^3 \log n)$ time as follows. Let the root of the ring separator tree be u . Given a query point q , check its distance to p_u . If $d_{\mathcal{M}}(q, p_u) \leq r_u$, then recurse on the left subtree. Otherwise, recurse on the right subtree. At the end, return the nearest point to q among p_v , where v is on the path traversed by the algorithm.

The running time of this procedure is clearly dominated by the height of the tree which is $O(\lambda^3 \log n)$.

To see that this is indeed $2n$ -ANN, let a be the vertical path in the tree traversed by the algorithm, and let b be the vertical path in the tree connecting the root to the nearest neighbor of q in P . Let v be the lowest common vertex of a and b . Suppose that a continued on the left subtree of v while b continued on the right subtree. In this case the distance from q to the nearest neighbor is at least $r_v/2n$, while $d_{\mathcal{M}}(p_v, q) \leq r_v$. Thus p_v is $2n$ -ANN.

If a continued on the right subtree of v while b continued on the left subtree of v , then the distance from the nearest neighbor is at least $r_v/2n + (d_{\mathcal{M}}(p_v, q) - r_v)$, while p_v is at distance $d_{\mathcal{M}}(p_v, q)$. The ratio between these two quantities is clearly at most $2n$. \square

Remark 1. As is pointed out in [29, 33], it is possible to duplicate points in the ring for the two subtrees. Hence we can actually partition the $\mathbf{b}(p, 2r) \setminus \mathbf{b}(p, r)$ into $t \leq n$ subrings and choose to duplicate a “light” ring. When $t = 1$, we obtain the ring separator tree from [33] that supports $O(1)$ -ANN queries, but requires $n^{2^{O(\dim)}}$ storage. For general $t \leq n$ we obtain a data-structure that supports $O(t)$ -ANN queries, and that by choosing the right ring to duplicate, consumes only $n^{(3 \log 2\lambda)^{1/t}}$ storage. To see this, we set $\beta = (3 \log 2\lambda)^{1/t}$ and prove by induction on n that it is possible to find a ring such that the number of leaves in the tree is at most n^β . Denote $\eta_i = |\mathbf{b}(p, (1 + i/t)r)|/n$. Note that $(2\lambda)^{-3} \leq \eta_0 \leq \eta_1 \leq \dots \eta_t \leq n/2$, and therefore there exists $i \leq t$ for which $\eta_{i-1} \geq \eta_i^\beta$; otherwise $(2\lambda)^{-3} < \eta_0^{\beta^t} \leq (1/2)^{\beta^t}$ which is a contradiction. Thus by duplicating the i th ring, and by applying the inductive hypothesis on the number of leaves in the subtrees, the resulting tree will have at most $(\eta_i n)^\beta + ((1 - \eta_{i-1})n)^\beta \leq (\eta_{i-1} + (1 - \eta_{i-1}))n^\beta$ leaves.

Thus, setting $t = O(\log \log \lambda \cdot \log n)$, we obtain a linear space ring separator tree that supports $O(t)$ -ANN queries in $O(\log n)$ time.

4.3. ANN algorithm for arbitrary spread. The algorithm for arbitrary spread is now pretty clear. During the preprocessing we construct the augmented net-tree from section 3. We also construct the low quality ring separator tree. The construction time is $2^{O(\dim)} n \log n$, and the space used is $2^{O(\dim)} n$.

Given a query point $q \in \mathcal{M}$, and the approximation parameter $\varepsilon > 0$, the query algorithm consists of the following three steps:

1. First, find $2n$ -ANN p_1 using the low quality ring separator tree of section 4.2.
2. Next, find a vertex u in the net-tree that is an ancestor for p_1 and that satisfies

$$\ell(\bar{p}(u)) - 1 \geq \lceil \log_\tau(16 \cdot d_{\mathcal{M}}(p_1, q)) \rceil \geq \ell(u).$$

Hence

$$d_{\mathcal{M}}(\text{rep}_u, q) \leq d_{\mathcal{M}}(\text{rep}_u, p_1) + d_{\mathcal{M}}(p_1, q) \leq 2.5 \cdot \tau^{\ell(u)} + \frac{1}{16} \tau^{\ell(\bar{p}(u))-1}.$$

3. We now split the analysis into two cases as follows:
 - (a) If $2.5 \cdot \tau^{\ell(u)} \geq \frac{1}{16} \tau^{\ell(\bar{p}(u))-1}$, then clearly $d_{\mathcal{M}}(\text{rep}_u, q) \leq 5\tau^{\ell(u)}$, and thus u satisfies the conditions of Lemma 4.1.
 - (b) If, on the other hand, $2.5 \cdot \tau^{\ell(u)} < \frac{1}{16} \tau^{\ell(\bar{p}(u))-1}$, then the packing property of the net-tree implies that

$$\begin{aligned} P \cap \mathbf{b}(q, d_{\mathcal{M}}(q, \text{rep}_u)) &\subset P \cap \mathbf{b}(\text{rep}_u, 2d_{\mathcal{M}}(q, \text{rep}_u)) \\ &\subset P \cap \mathbf{b}\left(\text{rep}_u, \frac{1}{4} \cdot \tau^{\ell(\bar{p}(u))-1}\right) \subset P_u, \end{aligned}$$

and therefore $\hat{p} \in P_u$. Thus, in this case u also satisfies the conditions of Lemma 4.1.

4. Set $l = \ell(u)$. Using the notation of Lemma 4.1, the fact that p_1 is a $2n$ -ANN implies that $f \geq l - (1 + \log n)$, thus by setting the number of steps to $t = \lceil \log(n/\varepsilon) \rceil$, and applying the algorithm of Lemma 4.1, we obtain $(1 + \varepsilon)$ -ANN.

The running time of the query is

$$\lambda^{O(1)} \log n + O(\log n) + \lambda^{O(1)} \log n + \varepsilon^{-O(\dim)} \leq \lambda^{O(1)} \log n + \varepsilon^{-O(\dim)}.$$

We summarize as follows.

THEOREM 4.3. *Given a set P of n points of bounded doubling dimension \dim in a metric space \mathcal{M} , one can construct a data-structure for answering ANN queries (where the quality parameter ε is provided together with the query). The query time is $2^{O(\dim)} \log n + \varepsilon^{-O(\dim)}$, the expected preprocessing time is $2^{O(\dim)} n \log n$, and the space used is $2^{O(\dim)} n$.*

Theorem 4.3 compares quite favorably with the result of Krauthgamer and Lee [33], which solves the same problem with the same (tight) query time but uses $O(2^{O(\dim)} n^2 \text{polylog}(n))$ space.

5. Fast construction of WSPD and spanners. Let P be an n -point subset of a metric space \mathcal{M} with doubling dimension \dim and a parameter $1/4 > \varepsilon > 0$. Denote by $A \otimes B$ the set $\{\{x, y\} \mid x \in A, y \in B\}$. A WSPD with parameter ε^{-1} of P is a set of pairs $\{\{A_1, B_1\}, \dots, \{A_s, B_s\}\}$ such that

1. $A_i, B_i \subset P$ for every i .
2. $A_i \cap B_i = \emptyset$ for every i .
3. $\cup_{i=1}^s A_i \otimes B_i = P \otimes P$.
4. $d_{\mathcal{M}}(A_i, B_i) \geq \varepsilon^{-1} \cdot \max\{\text{diam}(A_i), \text{diam}(B_i)\}$.

The notion of WSPD was defined by Callahan and Kosaraju [11] for Euclidean spaces. Talwar [44] have shown that this notion transfers to constant doubling metrics. In particular, he proves that any n -point metric with doubling dimension \dim admits WSPD in which the number of pairs is $n\varepsilon^{-O(\dim)} \log \Phi$. We improve this result.

LEMMA 5.1. For $1 \geq \varepsilon > 0$, one can construct an ε^{-1} -WSPD of size $n\varepsilon^{-O(\dim)}$, and the expected construction time is $2^{O(\dim)}n \log n + n\varepsilon^{-O(\dim)}$.

Furthermore, the pairs of the WSPD correspond to (P_u, P_v) , where u, v are vertices of a net-tree of P , and for any pair (P_u, P_v) in WSPD, $\text{diam}(P_u), \text{diam}(P_v) \leq \varepsilon d_P(\text{rep}_u, \text{rep}_v)$.

Proof. We compute the net-tree T using Theorem 3.1. For concreteness of the WSPD, assume also that some weak linear order \preceq is defined on the vertices of T . The WSPD is constructed by calling $\text{genWSPD}(u_0, u_0)$, where u_0 is the root of the net-tree T , and $\text{genWSPD}(u, v)$ is defined recursively as follows:

```

genWSPD(u, v)
  Assume  $\ell(u) > \ell(v)$  or  $(\ell(u) = \ell(v)$  and  $u \preceq v)$ 
    (otherwise exchange  $u \leftrightarrow v$ ).
  If  $8 \frac{2\tau}{\tau-1} \cdot \tau^{\ell(u)} \leq \varepsilon \cdot d_{\mathcal{M}}(\text{rep}_u, \text{rep}_v)$ , then
    return  $\{\{u, v\}\}$ 
  else
    Denote by  $u_1, \dots, u_r$  the children of  $u$ 
    return  $\bigcup_{i=1}^r \text{genWSPD}(u_i, v)$ .
```

For any node $u \in T$, we have $\text{diam}(P_u) \leq 2 \frac{2\tau}{\tau-1} \cdot \tau^{\ell(u)}$ (see Definition 2.1). In particular, for every output pair $\{u, v\}$, it holds that

$$\begin{aligned} \max\{\text{diam}(P_u), \text{diam}(P_v)\} &\leq 2 \frac{2\tau}{\tau-1} \cdot \max\{\tau^{\ell(u)}, \tau^{\ell(v)}\} \leq \frac{\varepsilon}{4} d_P(\text{rep}_u, \text{rep}_v) \\ &\leq \frac{\varepsilon}{4} (d_P(P_u, P_v) + \text{diam}(P_u) + \text{diam}(P_v)), \end{aligned}$$

and so $\max\{\text{diam}(P_u), \text{diam}(P_v)\} \leq \frac{\varepsilon}{4(1-\varepsilon/2)} d_P(P_u, P_v) \leq \varepsilon d_P(P_u, P_v)$, since $\varepsilon \leq 1$. Similarly, for any $x \in P_u$ and $y \in P_v$, we have

$$d_P(\text{rep}_u, \text{rep}_v) \leq d_P(x, y) + \text{diam}(P_u) + \text{diam}(P_v) \leq (1 + \varepsilon) d_P(x, y).$$

One can verify that every pair of points is covered by a pair of subsets $\{P_u, P_v\}$ output by the genWSPD algorithm.

We are left to argue about the size of the output (the running time is clearly linear in the output size). Let $\{u, v\}$ be an output pair and assume that the call to $\text{genWSPD}(u, v)$ was issued by $\text{genWSPD}(u, \bar{p}(v))$. We charge this call to $\bar{p}(v)$, and we will prove that each vertex is charged at most $\varepsilon^{-O(\dim)}$ times.

Fix $v' \in T$. It is charged by pairs of the form $\{u, v\}$ in which $\bar{p}(v) = v'$, and which were issued inside $\text{genWSPD}(u, v')$. This implies that $\ell(\bar{p}(u)) \geq \ell(v') \geq \ell(u)$.

Since the pair (u, v') was not generated by genWSPD , we conclude that $d_P(\text{rep}_{v'}, \text{rep}_u) \leq (8 \frac{2\tau}{\tau-1} \cdot \tau^{\ell(v')})/\varepsilon$. The set

$$U = \left\{ w \mid \ell(\bar{p}(w)) \geq \ell(v') \geq \ell(w) \text{ and } d_P(\text{rep}_{v'}, \text{rep}_w) \leq 8 \frac{2\tau}{\varepsilon(\tau-1)} \cdot \tau^{\ell(v')} \right\}$$

contains u , and U is a subset of $\mathcal{N}_C(\ell(v'))$. By Proposition 2.2, for every $u_1, u_2 \in U$, if $u_1 \neq u_2$, then $d_P(P_{u_1}, P_{u_2}) \geq \tau^{\ell(v')-1}/4$. By the doubling property, we have $|U| \leq \varepsilon^{-O(\dim)}$. We therefore infer that v' can be charged only by pairs in $U \times C_{v'}$, where $C_{v'}$ is the set of children of v' . We conclude that v' might be charged at most $|U| \cdot |C_{v'}| \leq (2/\varepsilon)^{O(\dim)} = \varepsilon^{-O(\dim)}$ times. Thus, the total number of pairs generated by the algorithm is $n\varepsilon^{-O(\dim)}$. \square

5.1. Spanners.

DEFINITION 5.2. *A t -spanner of a finite metric space P is a weighted graph G whose vertices are the points of P , and for any $x, y \in P$,*

$$d_P(x, y) \leq d_G(x, y) \leq t \cdot d_P(x, y),$$

where d_G is the metric of the shortest path on G .

THEOREM 5.3. *Given an n -point metric P with doubling dimension \dim and parameter $1 \geq \varepsilon > 0$, one can compute a $(1 + \varepsilon)$ -spanner of P with $n\varepsilon^{-O(\dim)}$ edges, in $2^{O(\dim)}n \log n + n\varepsilon^{-O(\dim)}$ expected time.*

Proof. Let $c \geq 16$ be an arbitrary constant, and set $\delta = \varepsilon/c$. Compute a δ^{-1} WSPD decomposition using the algorithm of the previous section. For every pair $\{u, v\} \in \text{WSPD}$, add an edge between $\{\text{rep}_u, \text{rep}_v\}$ with weight $d_P(\text{rep}_u, \text{rep}_v)$. Let G be the resulting graph; clearly, the resulting shortest path metric d_G dominates the metric d_P .

The upper bound on the stretch is proved by induction on the length of pairs in the WSPD. Fix a pair $x, y \in P$; by our induction hypothesis, we have that for every pair $z, w \in P$ such that $d_P(z, w) < d_P(x, y)$, it holds that $d_G(z, w) \leq (1 + c\delta)d_P(z, w)$.

The pair x, y must appear in some pair $\{u, v\} \in \text{WSPD}$, where $x \in P_u$ and $y \in P_v$. Thus $d_P(\text{rep}_u, \text{rep}_v) \leq (1 + 2\delta)d_P(x, y)$ and $d_P(x, \text{rep}_u), d_{\mathcal{M}}(y, \text{rep}_v) \leq \delta d_{\mathcal{M}}(\text{rep}_u, \text{rep}_v)$ by Lemma 5.1. By the inductive hypothesis

$$\begin{aligned} d_G(x, y) &\leq d_G(x, \text{rep}_u) + d_G(\text{rep}_u, \text{rep}_v) + d_G(\text{rep}_v, y) \\ &\leq (1 + c\delta)d_P(x, \text{rep}_u) + d_P(\text{rep}_u, \text{rep}_v) + (1 + c\delta)d_P(\text{rep}_v, y) \\ &\leq 2(1 + c\delta) \cdot \delta \cdot d_P(\text{rep}_u, \text{rep}_v) + d_P(\text{rep}_u, \text{rep}_v) \\ &\leq (1 + 2\delta + 2c\delta^2)(1 + 2\delta)d_P(x, y) \\ &\leq (1 + \varepsilon)d_P(x, y), \end{aligned}$$

since $\delta c \leq \varepsilon \leq 1$ and $16\delta \leq 1$ and $c \geq 11$. \square

6. Compact representation scheme. A CRS of a finite metric space P is a “compact” data-structure that can answer distance queries for pairs of points. We measure the performance of a CRS using four parameters $(P, S, Q, \bar{\kappa})$, where P is the preprocessing time of the distance matrix, S is the space used by the CRS (in terms of words), Q is the query time, and $\bar{\kappa}$ is the approximation factor.

The distance matrix by itself is a $(P = O(1), S = O(n^2), Q = O(1), \bar{\kappa} = 1)$ -CRS. The ε^{-1} -WSPD as well as the $(1 + \varepsilon)$ -spanner are representations of $(1 + O(\varepsilon))$ -approximation of the metric that consumes only $\varepsilon^{-O(\dim)}n$ space. However, naively it takes $\Omega(n)$ time to answer approximate distance queries in these data-structures.

In this section, we obtain the following theorem.

THEOREM 6.1. *For any n point metric with doubling dimension \dim , the following exist:*

- (a) A ($P = 2^{O(\dim)}n \log^2 n + \varepsilon^{-O(\dim)}n, S = \varepsilon^{-O(\dim)}n, Q = 2^{O(\dim)}, \bar{\kappa} = 1 + \varepsilon$)-CRS.
- (b) A ($P = 2^{O(\dim)} \cdot \text{poly}(n) + \varepsilon^{-O(\dim)}n, S = \varepsilon^{-O(\dim)}n, Q = O(\dim), \bar{\kappa} = 1 + \varepsilon$)-CRS.

For general n -point metrics, Thorup and Zwick [45] obtained a $(kn^{1+1/k}, kn^{1+1/k}, O(k), 2k - 1)$ -CRS, where $k \in \mathbb{N}$ is a prescribed parameter. The trade-off between the

approximation and the space is essentially tight for general metrics. Closer in spirit to our setting, Gudmunsson et al. [21, 22] considered metrics that are t -approximated by Euclidean distances in \mathbb{R}^d , where both d and t are (possibly large) constants. They showed that such metrics have $(O(n \log n), O(n), O(1), 1 + \varepsilon)$ -CRS (the O notation here hides constants that depend on ε , d , and t). Our scheme strictly extends³ their result since metrics that are t -approximated by a set of points in the d -dimensional Euclidean space have doubling dimension at most $O(d \log(2t))$. We further discuss previous work on a special type of CRS, called *distance labeling*, in section 6.3.

Our scheme is naturally composed of two parts. In section 6.1 we show that by using the net-tree it is possible to convert an A -approximate CRS into a $(1 + \varepsilon)$ -approximate CRS in essentially $O(\log A)$ query time (and even $O(\log \log A)$ query time). We then show in section 6.2 how to obtain an $O(1)$ -approximate CRS using Assouad's embedding. In section 6.3 we observe that Assouad's embedding can be used in distance labeling schema.

6.1. Approximation boosting lemma. Assume we are given a data-structure \mathcal{A} , which is a $(P, S, Q, \bar{\kappa})$ -CRS of a set $P \subset \mathcal{M}$, where $\bar{\kappa} \leq 3n^2$. In this section, we derive a CRS with improved approximation. Besides storing the data-structure of \mathcal{A} , we also need the following data-structures:

1. The net-tree T , augmented so that it supports the following operations:
 - (a) $O(\log n)$ time access for ancestors of a given level as defined in section 3.5.
 - (b) Constant time access for an ancestor at a given level l of a given vertex x , when $l + 6 \log n$ is at least the level of a given ancestor z of x .
 - (c) A constant time access for the lca of two vertices in T [4].
2. An ε^{-1} -WSPD W on the net-tree T , with support for fast membership queries. For each pair we also store the distance between its representatives. By using hashing membership, queries can be answered in constant time.
3. The $(3n^2)$ -approximation HST H of section 3.2. The HST H should be augmented with the following features:
 - (a) A constant time access to lca queries, after a linear time preprocessing [4].
 - (b) Each vertex u of H contains pointers to the following set of vertices in T :

$$K_u = \{x \in T : d_{\mathcal{M}}(\text{rep}_x, \text{rep}_u) \leq 4\Delta_u \text{ and } \ell(x) < \log \Delta_u \leq \ell(\bar{p}(x))\}.$$

Note that $|K_u| \leq \lambda^{O(1)}$, and computing all these sets can be accomplished in $\lambda^{O(1)} n \log n$ time by finding the level $\lceil \log \Delta_u \rceil$ ancestor z of rep_u in T in $O(\log n)$ time, and then scanning $\text{Rel}(z)$.

All these data-structures can be created in $2^{O(\dim)} n \log n + \varepsilon^{-O(\dim)} n$ time and $\varepsilon^{-O(\dim)} n$ space.

Assuming Query- $\mathcal{A}(x, y)$ returns a value η , such that $d_{\mathcal{M}}(x, y)/\bar{\kappa} \leq \eta \leq d_{\mathcal{M}}(x, y)$, the query algorithm is as follows:

³Caveat: They use a weaker model of computation.

```

Query- $\mathcal{B}(x, y \in P)$ 
   $z \leftarrow \text{lca}_H(x, y)$ .
   $u' \leftarrow$  ancestor of  $x$  in  $T$  among  $K_z$ ,  $v' \leftarrow$  ancestor of  $y$  in  $T$  among  $K_z$ .
   $\eta \leftarrow \text{Query-}\mathcal{A}(x, y)$ .
   $u_0 \leftarrow$  ancestor of  $x$  in level  $\lfloor \log(\varepsilon\eta) \rfloor$ ,  $v_0 \leftarrow$  ancestor of  $y$  in level  $\lfloor \log(\varepsilon\eta) \rfloor$ .
   $u \leftarrow u_0$ ,  $v \leftarrow v_0$ .
  while  $\{u, v\} \notin W$  do
    if  $\ell(\bar{p}(u)) < \ell(\bar{p}(v))$  or  $(\ell(\bar{p}(u)) = \ell(\bar{p}(v)) \text{ and } \bar{p}(v) \preceq \bar{p}(u))$  then
       $u \leftarrow \bar{p}(u)$ 
    else
       $v \leftarrow \bar{p}(v)$ .
  return  $d_{\mathcal{M}}(\text{rep}_u, \text{rep}_v)$ .
    
```

Implementation details. u' is found by scanning all vertices in K_z (there are only $\lambda^{O(1)}$ such vertices) and checking which one of them is an ancestor of x in T (ancestorship can be checked using the lca operation on T). Note that an ancestor of x must be contained in K_z , since $d_{\mathcal{M}}(\text{rep}_z, x) \leq \Delta_z$, and thus the ancestor of the level immediately below $\log \Delta_z$ must be in K_z . A similar thing happens with v' . Both η and Δ_z are $3n^2$ -approximations to $d_{\mathcal{M}}(x, y)$, and therefore $\ell(u') - \ell(u_0) \leq 4 \log n + 3$; hence u_0 can be accessed in constant time. The same goes for v_0 .

The following lemma is an immediate consequence of the way in which the WSPD algorithm works.

LEMMA 6.2. *For a pair $\{s, t\} \in W$ (the ε^{-1} -WSPD), and $\ell(s) \leq \ell(t)$, one of the following conditions must be satisfied:*

1. $\ell(s) \leq \ell(t) < \ell(\bar{p}(s))$ and $\frac{2\tau}{\tau-1} \cdot \tau^{\ell(\bar{p}(s))} > \varepsilon \cdot d_{\mathcal{M}}(\text{rep}_{\bar{p}(s)}, \text{rep}_t)$, and $\frac{2\tau}{\tau-1} \cdot \tau^{\ell(s)} \leq \varepsilon \cdot d_{\mathcal{M}}(\text{rep}_s, \text{rep}_t)$.
2. $\ell(s) < \ell(t) = \ell(\bar{p}(s))$, and $\bar{p}(s) \preceq t$, and $\frac{2\tau}{\tau-1} \cdot \tau^{\ell(\bar{p}(s))} > \varepsilon \cdot d_{\mathcal{M}}(\text{rep}_{\bar{p}(s)}, \text{rep}_t)$.
 $\frac{2\tau}{\tau-1} \cdot \tau^{\ell(s)} \leq \varepsilon \cdot d_{\mathcal{M}}(\text{rep}_s, \text{rep}_t)$.

PROPOSITION 6.3. *The while loop finds a pair in W after $O(\log \bar{\kappa})$ steps.*

Proof. Denote by $\{u_0, v_0\}$ the pair with which the loop begins. It is straightforward to see that the loop climbs through all ancestor pairs $\{u, v\}$ of $\{u_0, v_0\}$ that satisfy either (i) $\ell(u) \leq \ell(v) < \ell(\bar{p}(u))$, or (ii) $\ell(u) < \ell(v) = \ell(\bar{p}(u))$ and $\bar{p}(u) \preceq v$.

Thus, if an ancestor pair exists in W , it will be found by the loop. As we argue in Lemma 5.1, there exists an ancestor pair $\{\bar{u}, \bar{v}\}$ of $\{x, y\}$ in W . Our choice $\{u_0, v_0\}$ ensures that u_0 is a descendant of \bar{u} at most $O(\log \bar{\kappa})$ levels down T , and the same goes for v_0 and \bar{v} . \square

Combining the above claims, implies the following.

LEMMA 6.4. *Let P be an n -point metric. Assume we are given a $(P, S, Q, \bar{\kappa})$ -CRS \mathcal{A} of a set P , where $\bar{\kappa} \leq 3n^2$. Then, one can obtain $(P', S', Q', 1 + \varepsilon)$ -CRS \mathcal{B} of P , where $P' = P + 2^{O(\dim)} n \log n + \varepsilon^{O(\dim)} n$, $S' = S + \varepsilon^{-O(\dim)} n$, $Q' = Q + O(\log \bar{\kappa})$.*

Remark 2. The dependence of the query time on $\bar{\kappa}$ can be improved from $O(\log \bar{\kappa})$ to $O(\log \log \bar{\kappa})$ without sacrificing any other parameter. The idea is to replace the “ladder climbing” in the algorithm above (the while loop) with a binary search on the $\log \bar{\kappa}$ levels. To do so we change the WSPD procedure to output *all* pairs it encounters. This clearly does not change asymptotically the size of W . We do a binary search on the $\log \bar{\kappa}$ relevant levels to find the lowest level pairs which still appear in the WSPD, and this gives the relevant pairs. We do not pursue this improvement rigorously, since in the CRS that we develop in the next section, the query time Q dominates $\bar{\kappa}$ anyway, and thus this would lead to no asymptotic savings in the query time.

6.2. Assouad embedding. To quickly obtain a constant approximation of the distance, we will use a theorem due to Assouad [2] (see also [27, 23]). The following is a variant of the original statement, tailored for our needs, and its proof is provided for the sake of completeness.

THEOREM 6.5. *Any metric space \mathcal{M} with doubling dimension \dim can be embedded in ℓ_∞^d , where $d \leq \varepsilon^{-O(\dim)}$, such that the metric $(\mathcal{M}, \sqrt{d_{\mathcal{M}}})$ is distorted by a factor of $1 + \varepsilon$.*

Proof. Fix $r > 0$. We begin by constructing an embedding $\phi^{(r)} : \mathcal{M} \rightarrow \mathbb{R}^{d_1}$, where $d_1 = \varepsilon^{-O(\dim)}$ with the following properties for every $x, y \in \mathcal{M}$:

1. $\|\phi^{(r)}(x) - \phi^{(r)}(y)\|_\infty \leq \min\{r, d_{\mathcal{M}}(x, y)\}$.
2. If $d_{\mathcal{M}}(x, y) \in [(1 + \varepsilon)r, 2r]$, then $\|\phi^{(r)}(x) - \phi^{(r)}(y)\|_\infty \geq (1 - \varepsilon)r$.

We take an εr -net $\mathcal{N}^{(r)}$ of \mathcal{M} and color it such that every pair $x, y \in \mathcal{N}^{(r)}$ for which $d_{\mathcal{M}}(x, y) \leq 4r$ is colored differently. Clearly, $d_1 = \varepsilon^{-O(\dim)}$ colors suffice. Associate with every color i a coordinate, and for $x \in \mathcal{M}$ define $\phi_i^{(r)}(x) = \max\{0, r - d_{\mathcal{M}}(x, C_i)\}$, where $C_i \subset \mathcal{N}^{(r)}$ is the set of points of color i .

We next check that the two properties above are satisfied. As $\phi_i^{(r)}(x) \in [0, r]$, it is clear that $|\phi_i^{(r)}(x) - \phi_i^{(r)}(y)| \leq r$ for every color i . The 1-Lipschitz property easily follows from the triangle inequality.

Next, assume that $d_{\mathcal{M}}(x, y) \in [(1 + \varepsilon)r, 2r]$. Since $d_{\mathcal{M}}(x, \mathcal{N}^{(r)}) \leq \varepsilon r$, there exists a color i for which $d_{\mathcal{M}}(x, C_i) \leq \varepsilon r$. This implies (by the triangle inequality) that $d_{\mathcal{M}}(y, C_i) \geq r$, and hence $|\phi_i^{(r)}(x) - \phi_i^{(r)}(y)| \geq (1 - \varepsilon)r$. Thus, the concatenation of all these coordinates, $\phi^{(r)} = \oplus_i \phi_i^{(r)}$, satisfies the condition above.

Let $d_2 = 8\varepsilon^{-1} \log(\varepsilon^{-1})$. The final embedding $\phi : \mathcal{M} \rightarrow \mathbb{R}^{d_2 d_1}$ is done by combining a weighted sum of $\phi^{(r)}$ as follows. Let $M_l(x)$ denote the matrix of size $d_2 \times d_1$, such that it is all zero, except the $(l \pmod{d_2})$ th row, which is $M_l(x) = \phi^{((1+\varepsilon)^l)}(x)$. Then

$$\phi(x) = \sum_{l \in \mathbb{Z}} \frac{M_l(x)}{(1 + \varepsilon)^{l/2}}.$$

To see that the embedding is a $1 + O(\varepsilon)$ -approximation of $\sqrt{d_{\mathcal{M}}}$, fix a pair of points $x, y \in \mathcal{M}$, and let $l_0 \in \mathbb{Z}$ such that $d_{\mathcal{M}}(x, y) \in [(1 + \varepsilon)^{l_0+1}, (1 + \varepsilon)^{l_0+2})$. Then in the relevant coordinates the ℓ_∞ distance between x and y is

$$\begin{aligned} \left\| \sum_{k \in \mathbb{Z}} (\psi_{l_0+d_2k}(x) - \psi_{l_0+d_2k}(y)) \right\|_\infty &\geq \|\psi_{l_0}(x) - \psi_{l_0}(y)\|_\infty - \sum_{k < 0} \|\psi_{l_0+d_2k}(x) - \psi_{l_0+d_2k}(y)\|_\infty \\ &\quad - \sum_{k > 0} \|\psi_{l_0+d_2k}(x) - \psi_{l_0+d_2k}(y)\|_\infty \\ &\geq (1 - \varepsilon)(1 + \varepsilon)^{l_0/2} - \sum_{k < 0} \frac{(1 + \varepsilon)^{2+l_0+d_2k}}{(1 + \varepsilon)^{(l_0+d_2k)/2}} - \sum_{k > 0} \frac{(1 + \varepsilon)^{2+l_0}}{(1 + \varepsilon)^{(l_0+d_2k)/2}} \\ &\geq (1 - \varepsilon) \cdot (1 + \varepsilon)^{l_0/2} - \varepsilon \cdot (1 + \varepsilon)^{l_0/2} - \varepsilon \cdot (1 + \varepsilon)^{l_0/2} \geq (1 - O(\varepsilon))\sqrt{d_{\mathcal{M}}(x, y)}. \end{aligned}$$

On the other hand, for each $j \in \{0, \dots, d_2 - 1\}$,

$$\begin{aligned} & \left\| \sum_{k \in \mathbb{Z}} (\psi_{l_0+j+d_2k}(x) - \psi_{l_0+j+d_2k}(y)) \right\|_{\infty} \\ & \leq \sum_{k \leq 0} \|\psi_{l_0+j+d_2k}(x) - \psi_{l_0+j+d_2k}(y)\|_{\infty} + \sum_{k > 0} \|\psi_{l_0+j+d_2k}(x) - \psi_{l_0+j+d_2k}(y)\|_{\infty} \\ & \leq \sum_{k \leq 0} \frac{(1 + \varepsilon)^{2+l_0+j+d_2k}}{(1 + \varepsilon)^{(l_0+j+d_2k)/2}} + \sum_{k > 0} \frac{(1 + \varepsilon)^{2+l_0}}{(1 + \varepsilon)^{(l_0+j+d_2k)/2}} = (1 + O(\varepsilon))\sqrt{d_{\mathcal{M}}(x, y)}. \end{aligned}$$

Hence $\|\phi(x) - \phi(y)\|_{\infty}$ is a $1 + O(\varepsilon)$ -approximation to $\sqrt{d_{\mathcal{M}}(x, y)}$. \square

The relevance of Assouad’s embedding to compact representations is clear: Intuitively, $\phi(x)$ is short, and given $\phi(x)$ and $\phi(y)$, we can compute the square of the ℓ_{∞} norm of the difference and obtain a $(1 + \varepsilon)$ -approximation to $d_{\mathcal{M}}(x, y)$. Note, however, that in order to be able to do this, we need to store $\Theta(\log(\Phi/\varepsilon))$ bits for each real number, which may require many words to be represented in our computation model (see section 2). We solve this issue in Lemma 6.7 by reducing the problem for metrics with arbitrary spread to a set of similar problems on metrics with only polynomial spread, on which Assouad’s embedding can be applied.

LEMMA 6.6. *Given an n -point metric M with a polynomially bounded spread Φ and doubling dimension dim , an Assouad embedding (with parameter ε) of M can be computed in $\varepsilon^{-O(\text{dim})}n \log^2 n$ time.*

Proof. We follow closely the proof of Theorem 6.5. For each scale $(1 + \varepsilon)^l$, we find in $O(n)$ time an $\varepsilon(1 + \varepsilon)^l$ -net $\mathcal{N}^{((1+\varepsilon)^l)}$ from the net-tree. We define a graph on this net: two points are connected by an edge if they are at a distance of at most $4(1 + \varepsilon)^l$. This can be done in $\varepsilon^{-O(\text{dim})}n$ time using a variant of $\text{Rel}()$ sets (basically, we compute sets like $\text{Rel}()$ that contain points at a distance of at most $O(\varepsilon^{-1})$ times the current scale, instead of 13 times the current scale). We then partition $\mathcal{N}^{((1+\varepsilon)^l)}$ to color-classes using the greedy algorithm. Implemented with hashing, it works in expected $O(n)$ steps. Next, for each color-class we construct a $(1 + \varepsilon/2)$ -ANN data-structure, and thus we can compute a $(1 + \varepsilon/2)$ -approximation to $d_{\mathcal{M}}(x, C_i)$. Note that in the proof of Theorem 6.5, by enlarging the constants a little bit, a $(1 + \varepsilon/2)$ -approximation suffices. We repeat this construction for the $\log_{1+\varepsilon} \Phi$ levels in the metric. The rest of the embedding calculation is straightforward.

The running time of the algorithm is therefore $\varepsilon^{-O(\text{dim})}n \log n \log \Phi$. \square

Remark 3. We believe that for $\varepsilon = 100$, a similar embedding can be constructed directly on the net-tree in $2^{O(\text{dim})}n$ time. The construction seems, however, much more complicated than the one described in Lemma 6.6. We have therefore decided that the slight gain in preprocessing time (overall, a factor of $\log n$, since the running time for constructing the net-tree is $2^{O(\text{dim})}n \log n$) is not worth the complications.

LEMMA 6.7. *If there exists a $(P, S, Q, \bar{\kappa})$ -CRS \mathcal{A} for every n -point metric with doubling dimension dim and spread $\leq 3(n/\varepsilon)^{12}$, and if P is concave, then there exists a $(P(4n) + 2^{O(\text{dim})}n \log n, S + O(n), Q + O(1), (1 + \varepsilon)\bar{\kappa})$ -CRS \mathcal{B} for every n -point metric with doubling dimension dim (without any assumption on the spread).*

Proof. Denote by H the low quality HST of section 3.2 which is a $3n^2$ -approximation to the given metric \mathcal{M} .

Set $a_1 = 0$ and $a_2 = \lceil 5(\log(\varepsilon^{-1}) + \log_2 n) \rceil$. Apply the following procedure on H to obtain two HSTs, H_1 and H_2 . Scan H from the top down. Retain the root, the leaves, and all internal vertices $u \in H$ with the following property: There exists $b > 0$ such that $\log_2 b \equiv a_i \pmod{\lceil 10(\log(\varepsilon^{-1}) + \log_2 n) \rceil}$ and $\Delta_{\bar{p}(u)} > b \geq \Delta_u$. The HST H_i is constructed naturally on the retained vertices: A retained vertex u is connected

to a parent v in H_i if v is the lowest retained ancestor of u in H .

Next, for each nonleaf vertex $u \in H_i$, $i \in \{1, 2\}$, denote by $C(u)$ the set of children of u . We observe that $R(C(u)) = \{\text{rep}_u \mid u \in C(u)\}$ has a spread of at most $3(n/\varepsilon)^{12}$. To see this, note that $\text{diam}(R(C(u))) \leq \Delta_u$, and, on the other hand, let b the largest real number such that $b < \Delta_u$ and $\log b \equiv a_i \pmod{\lceil 10(\log(\varepsilon^{-1}) \log_2 n) \rceil}$. Obviously $b \geq \Delta_u/(n/\varepsilon)^{10}$ and for every $x, y \in C(u)$, $\Delta_{\text{lca}_H(x,y)} \geq b$, and therefore $d_{\mathcal{M}}(x, y) \geq b/(3n^2)$. Thus, for each internal vertex $u \in H_i$ we can construct a $\bar{\kappa}$ -approximate CRS \mathcal{A} to $R(C(u))$. The whole construction time is therefore $2^{O(\text{dim})}n \log n + \sum_k P(n_k) \leq 2^{O(\text{dim})}n \log n + P(4n)$.

We equip H , H_1 , and H_2 with a data-structure for handling queries for lca and finding an ancestor at a given depth, both in constant time.

A distance query for the pair $x, y \in \mathcal{M}$ is processed as follows. Let $u_i = \text{lca}_{H_i}(x, y)$. let x_i be a child of u_i which is an ancestor of x in H_i , and similarly y_i . Note that u_i, x_i, y_i can be computed in constant time using the lca and depth ancestor queries.

Further observe that $\exists i \in \{1, 2\}$ for which $\max\{\Delta_{x_i}, \Delta_{y_i}\} \leq \Delta_{\text{lca}_H(x,y)}/(n/\varepsilon)^5$, and finding this i is an easy task.

We next query the CRS \mathcal{A} of $R(C(u_i))$ for an approximation of $d_{\mathcal{M}}(\text{rep}_{x_i}, \text{rep}_{y_i})$. From the above we deduce that

$$\max\{d_{\mathcal{M}}(x, \text{rep}_{x_i}), d_{\mathcal{M}}(y, \text{rep}_{y_i})\} \leq \frac{3\varepsilon^5}{n^3} \cdot \frac{\Delta_{\text{lca}_H(x,y)}}{3n^2} \leq \frac{3\varepsilon^5}{n^3} d_{\mathcal{M}}(x, y),$$

and therefore we have obtained a $\bar{\kappa}(1 + \varepsilon)$ -approximation to $d_{\mathcal{M}}(x, y)$. \square

COROLLARY 6.8. *Every n -point metric with doubling dimension dim has a $(P = \varepsilon^{-O(\text{dim})}n \log^2 n, S = \varepsilon^{-O(\text{dim})}n, Q = \varepsilon^{-O(\text{dim})}, \bar{\kappa} = 1 + \varepsilon)$ -CRS.*

Proof. For the proof, combine Lemmas 6.7 and 6.6. \square

Note that in Corollary 6.8 the query time depends on ε , in contrast to the claim in Theorem 6.1(a). This can be remedied using Lemma 6.4.

Proof of Theorem 6.1(a). Use the CRS of Corollary 6.8 with constant $\varepsilon_0 = 0.1$ as the bootstrapping CRS in Lemma 6.4. \square

Proof of Theorem 6.1(b). In [23], an alternative proof for the Assouad theorem is given with a much improved bound on the dimension of the host space: They prove that for any metric $(\mathcal{M}, d_{\mathcal{M}})$ with doubling dimension dim , it is possible to embed $(\mathcal{M}, d_{\mathcal{M}}^{1/2})$ in $\ell_{\infty}^{O(\text{dim})}$ with distortion $O(\text{dim}^2)$.⁴

This embedding can be done in polynomial time. Using it as a replacement for Lemma 6.6, we therefore obtain the claimed CRS. \square

Remark 4. The distortion of embedding into $\text{poly}(\text{dim})$ dimensional normed space cannot be improved below 1.9, since such an embedding gives a 1.9 approximate CRS which uses only $O(n \text{poly}(\text{dim}) \log \phi)$ bits of storage with label length which are polynomially dependent on dim (see section 6.3), but Talwar [44] have shown that such a CRS necessarily uses at least $n2^{\Omega(\text{dim})}$ bits, which is impossible for $\text{dim} = \Omega(\log \log n)$. In this sense the embedding technique of [23] cannot replace Assouad’s original technique.

It is still open whether the construction time in Theorem 6.1(b) can be improved to near linear. The difficulty lies in the algorithmic version of the Lovász local lemma. As discussed in Remark 2, distortions as high as $2^{2^{O(\text{dim})}}$ are tolerable in this context.

⁴If one wants to optimize the distortion using their technique, then it is possible to obtain $O(\text{dim})$ distortion when embedding into $\ell_p^{O(\text{dim} \log \text{dim})}$.

6.2.1. Lower bound. We next argue that beating the $\Omega(\dim)$ query time using schema similar to the one presented above is unlikely.

For given reals $d_1, D, d_2 > 1$, we say that a (d_1, D, d_2) -Assouad-type-scheme (ATS) exists if there is a monotone increasing bijection $f : [0, \infty) \rightarrow [0, \infty)$, such that for all finite metric spaces $(P, d_{\mathcal{M}})$, with doubling dimension at most d_1 , there exists $\phi : P \rightarrow \mathbb{R}_{\|\cdot\|_X}^{d_2}$, such that for $x, y \in P$, we have

$$\frac{d_{\mathcal{M}}(x, y)}{D} \leq f(\|\phi(x) - \phi(y)\|_X) \leq d_{\mathcal{M}}(x, y).$$

For example, the embedding of [23] cited above is a $(d_1, O(d_1^2), O(d_1))$ -ATS for any $d_1 > 1$, and it uses $f(x) = x^2$.

PROPOSITION 6.9. *If $d_2 \leq d_1/5$, then for any $D > 1$, no (d_1, D, d_2) -ATS exists.*

Proof. The argument distinguishes between two essential cases: ‘‘Concave’’ function f cannot be used in any ATS since it causes a violation of the triangle inequality. For ‘‘convex’’ functions f we slightly generalize an argument from [9] that uses topological considerations (Borsuk–Ulam theorem) to conclude the impossibility.

Indeed, fix a (d_1, D, d_2) -ATS with a function f , where $d_2 \leq d_1/5$. Denote $g : [0, \infty) \rightarrow [0, \infty)$, where $g = f^{-1}$.

Suppose first that $\sup_{0 < a \leq b < \infty} \frac{g(b)/b}{g(a)/a} = \infty$ (‘‘concave f ’’). Fix a and b such that $0 < a < b < \infty$ and $\frac{g(b)/b}{g(a)/a} \geq 100D$. Let $n = \lceil 2Db/a \rceil$, and let P be the line metric on $\{0, \dots, n\}$ such that $d_{\mathcal{M}}(i, j) = a|i - j|$. By the assumption, there exists $\phi : P \rightarrow \mathbb{R}_{\|\cdot\|_X}^{d_1}$ such that $\|\phi(i) - \phi(i + 1)\|_X \leq g(d_{\mathcal{M}}(i, i + 1)) = g(a)$, while on the other hand,

$$g(b) \leq g\left(\frac{\lceil 2Db/a \rceil a}{D}\right) = g\left(\frac{d_{\mathcal{M}}(0, n)}{D}\right) \leq \|\phi(0) - \phi(n)\|_X,$$

since g is monotone increasing, as f is monotone increasing. Then by the triangle inequality,

$$g(b) \leq \|\phi(0) - \phi(n)\|_X \leq \sum_{i=1}^n \|\phi(i - 1) - \phi(i)\|_X \leq n g(a) \leq 4D \frac{b g(a)}{a},$$

which implies that $\frac{g(b)/b}{g(a)/a} \leq 4D$, which is a contradiction.

Next, assume that there exists $C > 1$ such that $\sup_{0 < a \leq b < \infty} \frac{g(b) \cdot a}{g(a) \cdot b} \leq C$ (‘‘convex f ’’). Then, for any $a \leq b$ we have $\frac{g(b)a}{Cb} \leq g(a)$. In particular, we have $\frac{g(d_{\mathcal{M}}(x, y))(d_{\mathcal{M}}(x, y)/D)}{C d_{\mathcal{M}}(x, y)} \leq g(d_{\mathcal{M}}(x, y)/D)$. Namely,

$$\frac{g(d_{\mathcal{M}}(x, y))}{C \cdot D} \leq g\left(\frac{d_{\mathcal{M}}(x, y)}{D}\right) \leq \|\phi(x) - \phi(y)\|_X \leq g(d_{\mathcal{M}}(x, y)).$$

Since $\|\cdot\|_X$ is d_2 -dimensional, by John’s theorem (see [3, Chap. V]) it can be approximated by $\|\cdot\|_2$ up to a $\sqrt{d_2}$ factor. We thus have a $C' > 1$ such that for any d_1 -dimensional finite metric $(P, d_{\mathcal{M}})$, there exists $\phi' : (P, d_{\mathcal{M}}) \rightarrow \mathfrak{H}_{\|\cdot\|_2}^{d_2}$ satisfying

$$(6.1) \quad \frac{g(d_{\mathcal{M}}(x, y))}{C'} \leq \|\phi'(x) - \phi'(y)\|_2 \leq g(d_{\mathcal{M}}(x, y)).$$

We next estimate how much $g \circ d_{\mathcal{M}}$ distorts $d_{\mathcal{M}}$ as a function of the spread of P . Assume that $\min_{x \neq y \in P} d_{\mathcal{M}}(x, y) = a_1$ and $\max_{x \neq y \in P} d_{\mathcal{M}}(x, y) = b_1$, that is, $\Phi(P) = b_1/a_1$. Then

$$\begin{aligned} \max_{a_1 \leq t} \frac{g(t)}{t} &= \frac{g(a_1)}{a_1} \cdot \max_{a_1 \leq t} \frac{g(t)a_1}{tg(a_1)} \leq C \frac{g(a_1)}{a_1}, \\ \max_{s \leq b_1} \frac{s}{g(s)} &= \frac{b_1}{g(b_1)} \cdot \max_{s \leq b_1} \frac{g(b_1)s}{b_1g(s)} \leq C \frac{b_1}{g(b_1)}. \end{aligned}$$

Thus, considering the “distortion” of g , we have

$$\frac{\max_{x \neq y \in P} \frac{g(d_{\mathcal{M}}(x, y))}{d_{\mathcal{M}}(x, y)} \cdot \max_{x \neq y \in P} \frac{d_{\mathcal{M}}(x, y)}{g(d_{\mathcal{M}}(x, y))}}{\Phi(P)} \leq \frac{C \frac{g(a_1)}{a_1} \cdot C \frac{b_1}{g(b_1)}}{b_1/a_1} = C^2 \frac{g(a_1)}{g(b_1)}.$$

As $g(0) = 0$ and $\lim_{x \rightarrow \infty} g(x) = \infty$, we conclude that this ratio tends to 0 as the spread $\Phi(P)$ tends to ∞ . Combining it with (6.1), we conclude that for $\widehat{\phi} : (P, d_{\mathcal{M}}) \rightarrow \mathfrak{R}_{\|\cdot\|_2}^{d_2}$, defined as $\widehat{\phi}(x) = \phi'(x)$, we have $\text{dist}(\widehat{\phi}) = o(\Phi(P))$. We will next show that this is impossible when P is a sufficiently dense net of \mathbb{S}^{d_2} .

Let $0 < \eta \leq 0.1$. We take $P = P_{\eta}$ to be an η -net of $\mathbb{S}_{\|\cdot\|_2}^{d_2}$. The finite metric P_{η} has doubling dimension at most d_1 . From the above we can embed $\phi' : P_{\eta} \rightarrow \mathfrak{R}_{\|\cdot\|_2}^{d_2}$ with distortion $o(\Phi(P)) = o(\eta^{-1})$. By scaling we may assume that this embedding is 1-Lipschitz. By Kirszbraun’s theorem (see [7, Chap. 1]), the embedding ϕ' can be extended to the whole sphere $\widehat{\phi}' : \mathbb{S}_{\|\cdot\|_2}^{d_2} \rightarrow \mathbb{R}_{\|\cdot\|_2}^{d_2}$ without increasing the Lipschitz constant. The Borsuk–Ulam theorem (cf. [37]) states that there exists $x \in \mathbb{S}^k$ such that $\widehat{\phi}'(x) = \widehat{\phi}'(-x)$. Note that $\exists y, z \in P_{\eta}$ such that $\|x - y\|_2 \leq \eta$, and $\|(-x) - z\|_2 \leq \eta$. Since $\widehat{\phi}'$ is 1-Lipschitz, we have

$$\|\phi'(y) - \phi'(z)\|_2 = \|\widehat{\phi}'(y) - \widehat{\phi}'(z)\|_2 \leq \|\widehat{\phi}'(y) - \widehat{\phi}'(x)\|_2 + \|\widehat{\phi}'(-x) - \widehat{\phi}'(z)\|_2 \leq 2\eta.$$

On the other hand, $\|y - z\|_2 \geq 1 - 2\eta$, which means that the Lipschitz constant of ϕ'^{-1} , and thus the distortion of ϕ' , is at least $\Omega(\eta^{-1})$. This is a contradiction when η is a sufficiently small positive number, since we argued above that the distortion must be $o(\Phi(P)) = o(\eta^{-1})$. \square

6.3. Distance labeling. An approximate distance labeling scheme (ADLS) seeks to compute for each point in the metric a short label such that given the labels of a pair of points, it is possible to compute efficiently an approximation of the pairwise distance. Thus, ADLS is a stricter notion of compact representation.⁵ This notion was studied, for example, in [19, 18, 45].

In the constant doubling dimension setting, Gupta et al. [23] have shown a $(1 + \varepsilon)$ -embedding of the metric in $\ell_{\infty}^{O(\log n)}$. This implies a $(1 + \varepsilon)$ -ADLS with $O(\log n \log \Phi)$ bits for each label (the O notation here hides constants that depend on ε and dim). Talwar [44] has shown an improved $(1 + \varepsilon)$ -ADLS with only $\varepsilon^{-O(\text{dim})} \log \Phi$ bits per label. Slivkins [43] has shown a $(1 + \varepsilon)$ -ADLS with $\varepsilon^{-O(\text{dim})} \log^2 n \log \log \Phi$ bits per label. Their techniques seem to be very different from each other.

⁵When comparing the storage of ADLSs to that of the CRSs from the previous sections, note that here we count *bits*, whereas in the rest of the paper we count *words* of length $O(\log n + \log \log \Phi + \log \varepsilon^{-1})$.

Here we improve Slivkins’ result and unify it with Talwar’s result under the same technique.

PROPOSITION 6.10. *Given a finite metric space, one can build a $(1 + \varepsilon)$ -ADLS with*

$$\min \left\{ \varepsilon^{-O(\dim)} \log \Phi, \varepsilon^{-O(\dim)} \log n(\log n + \log \log \Phi) \right\}$$

bits per label.

Furthermore, there exist one-dimensional finite metric spaces of size n , and spread $\Phi \geq 2^{2n}$ for which any 1.9-ADLS requires labels of size $\Omega(\log n \log \log \Phi)$ bits per label.

Proof (sketch). First, labels of length $\varepsilon^{-O(\dim)} \log \Phi$ follow directly from Theorem 6.5: We have $\varepsilon^{-O(\dim)}$ coordinates, and, as discussed after the proof of Theorem 6.5, we need only $O(\log(\Phi/\varepsilon))$ bits of accuracy for each coordinate.

We next show a $(1 + \varepsilon)$ -ADLS using $\varepsilon^{-O(\dim)} \log n(\log n + \log \log \Phi)$ bits per label. We do so by presenting a “distributed implementation” of the data-structure used to prove Corollary 6.8. That data-structure consists of two trees (HSTs) H_1, H_2 on the same set of leaves: the points of the metric. Given two points x^1, x^2 , we compute $u_i = \text{lca}_{H_i}(x^1, x^2)$, and x_i^j is the ancestor of x^j in H_i which is the child of u_i . We then apply an Assouad embedding $A(x_i^j)$ that uses $O(\log n + \log \log \Phi + \log(\varepsilon^{-1}))$ bits. We define an identifier $I(v)$ of vertex $v \in H_i$ to be $A(v)$ concatenated with the Δ_v (encoded with $O(\log \log \Phi)$ bits). Hence, given two points x^1, x^2 , using the identifiers $I(x_1^1), I(x_1^2), I(x_2^1), I(x_2^2), I(u_1), I(u_2)$, we can compute a $(1 + \varepsilon)$ -approximation of $d_{\mathcal{M}}(x_1, x_2)$. We now use (the proof of) a result of Peleg [39]: Given an n -vertex rooted tree with identifiers $I(v)$ of maximum length s on the vertices, it is possible to efficiently compute labels $L(v)$ of length $O(\log n(\log n + s))$ to the vertices, such that given $L(x)$ and $L(y)$ one can efficiently decode $I(u)$, where $u = \text{lca}(x, y)$.

Unfortunately, we need a little bit more, namely, access to the children of u which are the ancestors of x and y . In order to obtain it we tinker with the construction of Peleg: In his Definition 3.2 from [39], we extend the tuple $Q_i(v)$ to be

$$Q_i(v) = \left\langle \langle i - 1, I(\gamma_{i-1}(v)) \rangle, \langle i, I(\gamma_i(v)) \rangle, \langle i + 1, I(\gamma_{i+1}(v)) \rangle, \underline{\langle i, I(\text{hs}(\gamma_i(v))) \rangle} \right\rangle,$$

where $\text{hs}(u)$ is the *heavy sibling* of u (the underlined part is our extension). By studying Peleg’s construction, it is easy to verify that this extension suffices.

The above construction is asymptotically optimal in terms of n and Φ when $\Phi \geq 2^{2n}$, as we now prove. In [19] there is a family of n -vertex weighted rooted binary trees, such that any exact distance labeling scheme of the *leaves* requires labels of length $\Omega(\log n \log M)$ bits, where the edge weight is in the range $\{0, \dots, M - 1\}$. A further property of that family of trees is that the depth $h = M \log_2 n$ (i.e., the distance from the root) of all the leaves is the same. We next transform each tree T in that family into an HST H by giving every vertex v a label $2^{-\text{depth}_T(v)}$. For any two leaves x and y , let $d_T(x, y) = 2(h + \log_2 d_H(x, y))$. Furthermore, even a 1.9 approximation of $d_H(x, y)$ allows us to recover the exact value of $d_H(x, y)$, since this value is an integral power of 2. Let us summarize: Given a 1.9 approximation of the distance in H allows us to obtain the exact distance in T . Therefore by setting $M = (\log_2 \Phi)/n$, it proves a lower bound of $\Omega(\log n \log \log \Phi)$ on the average label’s length for a 1.9-ADLS for this family of HSTs. Since these HSTs are binary, their doubling dimension is 1. \square

After a preliminary version of this paper appeared, Slivkins [42] managed to produce an ADLS with labels of length $\varepsilon^{-O(\dim)} \log n \log \log \Phi$, which improves upon our construction in the range $n^{\log \log n} \ll \Phi \ll 2^n$.

7. Doubling measure. A measure μ on a metric space \mathcal{M} is called η -doubling if, for any $x \in \mathcal{M}$ and $r \geq 0$, $\mu(\mathbf{b}(x, 2r)) \leq \eta \cdot \mu(\mathbf{b}(x, r))$. *Doubling measure* is already a useful notion in the analysis of metric spaces (see [27]) and has recently been used in some algorithmic applications [43]. Vol'berg and Konyagin [47] proved that any compact λ -doubling metric space has a $\lambda^{O(1)}$ -doubling measure. Wu's proof of this theorem [48] can be implemented in linear time on the net-tree.

We assume that the net-tree T is already given. Denote by $\deg(v)$ the number of children of $v \in T$. Let $\gamma = \max_{v \in T} \deg(v)$ be the maximum degree in T . As we have seen before, $\gamma \leq 2^{O(\dim)}$. The probability measure μ is computed by calling $\text{Partition}(\text{root}, 1)$, where Partition is defined recursively as follows.

```

Partition( $u \in T, p_u \in [0, 1]$ ).
  if  $u$  is a leaf then
    Set  $\mu(\{\text{rep}_u\}) \leftarrow p_u$ .
  else
    for each child  $v$  of  $u$  with  $\text{rep}_v \neq \text{rep}_u$  do
      Set  $p_v \leftarrow p_u / \gamma$ .
      Call Partition( $v, p_v$ ).
    Let  $v_0$  be the unique child of  $u$  such that  $\text{rep}_{v_0} = \text{rep}_u$ .
    Set  $p_{v_0} \leftarrow p_u(1 - (\deg(u) - 1) / \gamma)$ .
    Call Partition( $v_0, p_{v_0}$ ).
    
```

PROPOSITION 7.1. *For any $u \in T$, we have $p_u = \mu(P_u)$.*

Proof. The proof is by straightforward induction on the height of T . □

PROPOSITION 7.2. *Fix $l \in \mathbb{N}$, and two vertices u and v in T , such that $\max\{\ell(u), \ell(v)\} < l \leq \min\{\ell(\bar{p}(u)), \ell(\bar{p}(v))\}$ and $d_{\mathcal{M}}(\text{rep}_u, \text{rep}_v) \leq 40\tau^l$. Then $p_u \leq \gamma^{O(1)}p_v$.*

Proof. Denote $w = \text{lca}_T(u, v)$, and denote by $w = u_0, u_1, \dots, u_a = u$ the path in T from w to u , and by $w = v_0, v_1, \dots, v_b = v$ the path in T from w to v .

We claim that for any $i \geq 1$, if $\ell(u_i) > l + 3$, then $\text{rep}_{u_i} \neq \text{rep}_{u_{i+1}}$. Indeed, otherwise

$$\begin{aligned}
 d_{\mathcal{M}}(\text{rep}_{u_i}, \text{rep}_v) &\leq d_{\mathcal{M}}(\text{rep}_{u_{i+1}}, \text{rep}_u) + d_{\mathcal{M}}(\text{rep}_u, \text{rep}_v) \\
 &\leq \frac{2\tau}{\tau - 1} \cdot \tau^{\ell(u_{i+1})} + 40\tau^l \leq \frac{2}{\tau - 1} \tau^{\ell(u_i)} + 40\tau^{-4} \cdot \tau^{\ell(u_i)} \leq \frac{\tau^{\ell(u_i)}}{4},
 \end{aligned}$$

but this is a contradiction to the packing property of Definition 2.1, since $v \notin P_{u_i}$ (note that for this argument to work, τ needs to be a large enough constant, say 11).

Next, we claim that for any $i \geq 1$ for which $\ell(u_i) > l + 3$, $\ell(u_{i-1}) = \ell(u_i) + 1$. Otherwise, $\ell(u_{i-1}) - 1 \geq \ell(u_i) + 1$, implying

$$\begin{aligned}
 d_{\mathcal{M}}(\text{rep}_{u_i}, \text{rep}_v) &\leq d_{\mathcal{M}}(\text{rep}_{u_{i+1}}, \text{rep}_u) + d_{\mathcal{M}}(\text{rep}_u, \text{rep}_v) \leq \frac{2\tau}{\tau - 1} \cdot \tau^{\ell(u_i)} + 40\tau^{-4} \cdot \tau^{\ell(u_i)} \\
 &= \left(\frac{2\tau}{\tau(\tau - 1)} + \frac{40}{\tau^5} \right) \cdot \tau^{\ell(u_i)+1} \leq \frac{\tau^{\ell(u_{i-1})-1}}{4} = \frac{\tau^{\ell(\bar{p}(u_i))-1}}{4},
 \end{aligned}$$

contradicting the packing property of Definition 2.1, since $v \notin P_{u_i}$.

Thus, the path between u and w is full, containing vertices on all levels, except maybe the last three levels. Furthermore, the representatives are different in each level. We therefore conclude that $p_u \leq p_w/\gamma^{\ell(w)-l-4}$. On the other hand, $p_v \geq p_w\gamma^{\ell(w)-l+1}$. Therefore $p_u \leq \gamma^5 p_v$. \square

THEOREM 7.3. *For any n -point metric space having doubling dimension dim , it is possible to construct a $2^{O(\text{dim})}$ -doubling measure in $2^{O(\text{dim})}n \log n$ time.*

Proof. The running time of Partition is clearly linear and is dominated by the time to construct the net-tree.

We are left to prove that μ is a $\lambda^{O(1)}$ -doubling measure. Let $x \in P$ and $r > 0$. Denote $N = \{u \in T \mid \ell(u) \leq \log_\tau(r/8) < \ell(\bar{p}(u))\}$. As we have seen in Proposition 2.2, the representatives of the vertices of N forms a net in the right scale. In particular, there exists $\hat{x} \in N$ such that $d_{\mathcal{M}}(x, \text{rep}_{\hat{x}}) \leq 3r/8$ and $P_{\hat{x}} \subset \mathbf{b}(\text{rep}_{\hat{x}}, 3r/8) \subset \mathbf{b}(x, r)$. Hence $p_{\hat{x}} \leq \mu(\mathbf{b}(x, r))$. On the other hand, any two different representatives of vertices from N are at least $r/40$ separated, and therefore, for $X = N \cap \{u \in T \mid \text{rep}_u \in \mathbf{b}(x, 3r)\}$, we have $|X| \leq \lambda^{O(1)}$. Note that $\mathbf{b}(x, 2r) \subset \cup_{u \in X} P_u$, and therefore

$$\mu(\mathbf{b}(x, 2r)) \leq \sum_{u \in X} p_u \leq |X| \max_{u \in X} p_u.$$

By Proposition 7.2, $\max_{u \in X} p_u \leq \lambda^{O(1)} p_{\hat{x}}$. We conclude that $\mu(\mathbf{b}(x, 2r)) \leq \lambda^{O(1)} \mu(\mathbf{b}(x, r))$. \square

We note in passing that algorithm Partition can be programmed in our computational model since every point gets at least a $2^{-O(n \log n)}$ measure, which can be easily represented in a floating-point word of length $O(\log n)$. Moreover, the algorithm has a “built in” mechanism to handle rounding error: instead of dividing by γ , we can divide by, say, 2γ , and now rounding errors are automatically offset in the measure given to v_0 .

8. Lipschitz constant of mappings.

DEFINITION 8.1. *A function $f : (P, \nu) \rightarrow (M, \rho)$ is K -Lipschitz if for any $x, y \in P$, we have $\rho(f(x), f(y)) \leq K \cdot \nu(x, y)$.*

A point $x \in P$ is K -Lipschitz if, for any $y \in P$, we have $\rho(f(x), f(y)) \leq K \cdot \nu(x, y)$.

Thus, given a set of points $P \subseteq \mathbb{R}^d$, and a mapping $f : P \rightarrow \mathbb{R}^d$, it is natural to ask how quickly we can compute the Lipschitz constant for f on the set P , and more specifically, to compute it for every point of P .

8.1. The low-dimensional Euclidean case. Here, we consider a mapping $f : P \rightarrow (M, \rho)$, where $P \subseteq \mathbb{R}$ is of size n , and (M, ρ) is an arbitrary metric space given as a matrix.

PROPOSITION 8.2. *Computing the Lipschitz constant for f on P can be done in $O(n \log n)$ time.*

Proof. Indeed, let a, b, c be three numbers in P such that $a < b < c$. Observe that

$$\frac{\rho(f(c), f(a))}{c - a} \leq \frac{\rho(f(c), f(b)) + \rho(f(b), f(a))}{c - b + b - a} \leq \max\left(\frac{\rho(f(c), f(b))}{c - b}, \frac{\rho(f(b), f(a))}{b - a}\right),$$

since for any p, q, r, s positive numbers such that $p/q \leq r/s$, we have $p/q \leq (p + r)/(q + s) \leq r/s$. Thus, the Lipschitz constant is realized by a consecutive pair of points in P . We can therefore sort P and compute the slope for every consecutive pair. Clearly, the maximum is the Lipschitz constant of f . \square

PROPOSITION 8.3. *Let P be a set of n numbers on the real line, and let $f : P \rightarrow \mathfrak{R}$ be a given mapping. One can compute the Lipschitz constant of f on every point of P in $O(n \log^2 n)$ time.*

Proof. Consider the set $Q = \{(p, f(p)) \mid p \in P\}$. Let p be a point in P , and let L_p be the set of points of Q strictly to the left of p (according to the x -order), and let R_p be the set of points to its right. Denote by $\mathcal{CH}(A)$ the convex hull of $A \subset \mathfrak{R}^2$. If we know the tangents to $\mathcal{CH}(L_p)$ and $\mathcal{CH}(R_p)$ that pass through p , then we can compute the Lipschitz constant of p in constant time (i.e., it is the slope of the tangent with the largest slope).

Here, one can use the data-structure of Overmars and van Leeuwen [38], which supports the maintenance of the convex hull under insertions, deletions, and tangent queries in $O(\log^2 n)$ per operation. Indeed, sort the points of P from left to right. Let p_1, \dots, p_n be the sorted points. Clearly, given $\mathcal{CH}(L_{p_i})$ and $\mathcal{CH}(R_{p_i})$ stored in the dynamic convex hull data-structure, we can compute $\mathcal{CH}(L_{p_{i+1}})$ and $\mathcal{CH}(R_{p_{i+1}})$ by deleting p_{i+1} from $\mathcal{CH}(R_{p_i})$ and inserting p_i into $\mathcal{CH}(L_{p_i})$. Thus, we can compute all the relevant convex hulls in $O(n \log^2 n)$ time. Furthermore, when we have $\mathcal{CH}(L_{p_i})$ and $\mathcal{CH}(R_{p_i})$, we perform tangent queries to compute the Lipschitz constant of p_i . Thus, the overall running time is $O(n \log^2 n)$. \square

THEOREM 8.4. *Given a set P of n points in the plane, and a mapping $f : P \rightarrow \mathfrak{R}$, one can compute the Lipschitz constant of f in $O(n \log^2 n)$ expected time.*

Proof. Assume that we know that f is K -Lipschitz on a set $Q \subseteq P$, and we would like to verify that it is K -Lipschitz on $\{q\} \cup Q$, where $q \in P \setminus Q$. This can be visualized as follows: From every point $p \in P$, there is an associated point in \mathfrak{R}^3 , which is $\hat{p} = (p_x, p_y, f(p))$. Being K -Lipschitz, as far as p is concerned, implies that q must lie below the upper cone of slope K emanating from \hat{p} and above the lower cone of slope K emanating from \hat{p} . Thus, if we collect all those upper cones, then q must lie below their lower envelope. However, since the upper cones all have the same slope, their lower envelope is no more than a (scaled) version of an additive weighted Voronoi diagram in the plane. Such a diagram can be computed in $O(n \log n)$ time for n points, and a point-location query in it can be performed in $O(\log n)$ time.

In fact, using the standard Bentley–Saxe technique [6], one can build a data-structure, where one can insert such upper cones in $O(\log^2 n)$ amortized time, given a query point q in the plane, and decide in $O(\log^2 n)$ time which of the cones inserted lies on the lower envelope vertically above q . Similar data-structures can be built for the upper envelope of the lower cones.

Thus, if we conjecture that the Lipschitz constant is K , then we can verify it for P in $O(n \log^2 n)$ time by inserting the points of P into the upper and lower envelope data-structure described above. However, let us assume that K is too small. Then, after inserting a subset Q of points into the data-structure, we will try to verify that the Lipschitz constant for a point $p \in P$ is K and fail. Then, it must be that the Lipschitz constant of f on $Q \cup \{p\}$ is realized by p . Thus, we can compute the Lipschitz constant of p in $Q \cup \{p\}$ in $O(|Q|)$ time, update our guess K , and rebuild the upper and lower data-structures for $Q \cup \{p\}$.

Of course, in the worst case, this would required $O(n^2 \log^2 n)$ running time (i.e., we would fail on every point). However, it is well known that if we randomly permute the points, and handle the points according to this ordering, then the value of the Lipschitz constant on every prefix would change $O(\log n)$ times in expectation. Thus, this would lead to $O(n \log^3 n)$ expected running time. Moreover, a slightly more careful analysis shows that the expected running time is $O(n \log^2 n)$. See [16] for details of such analysis. \square

8.2. Constant doubling dimension to arbitrary metric.

THEOREM 8.5. *We are given a metric (P, ν) of n points having doubling dimension d , and a mapping $f : P \rightarrow (\mathcal{M}, \rho)$, where \mathcal{M} is an arbitrary metric space. Then one can compute a $(1 + \varepsilon)$ -approximation of the Lipschitz constant of f in $n\varepsilon^{-O(d)} \log^2 n$ expected time.*

Proof. The algorithm is as follows:

1. Compute ε^{-1} -WSPD of P according to section 5.
2. Set $K \leftarrow 0$.
3. For every pair $(A, B) \in \varepsilon^{-1}$ -WSPD do:
 - (a) Obtain *some* pair of points $a \in A$ and $b \in B$.
 - (b) Compute $K \leftarrow \max\{K, \frac{\rho(f(a), f(b))}{\nu(a, b)}\}$.

Obviously the value K computed by the algorithm above is not larger than the Lipschitz constant of f . We next show that it is not much smaller. Let $x, y \in P$ be a pair in which f obtains its Lipschitz constant, i.e., $\frac{\rho(f(x), f(y))}{\nu(x, y)} = \max_{a \neq b} \frac{\rho(f(a), f(b))}{\nu(a, b)}$. Let $\{A, B\} \in \text{WSPD}$ be a pair such that $x \in A, y \in B$. Our algorithm chooses some pair $a \in A, b \in B$. Using the triangle inequality we have

$$\begin{aligned} \frac{\rho(f(a), f(b))}{\nu(a, b)} &\geq \frac{\rho(f(x), f(y)) - \text{diam}(f(A)) - \text{diam}(f(B))}{\nu(x, y) + \text{diam}(A) + \text{diam}(B)} \\ &\geq \frac{\rho(f(x), f(y)) - \text{diam}(f(A)) - \text{diam}(f(B))}{(1 + 2\varepsilon)\nu(x, y)}. \end{aligned}$$

If $\max\{\text{diam}(f(A)), \text{diam}(f(B))\} \leq \varepsilon \cdot \rho(f(x), f(y))$ then we conclude that

$$\frac{\rho(f(a), f(b))}{\nu(a, b)} \geq \frac{(1 - 2\varepsilon)\rho(f(x), f(y))}{(1 + 2\varepsilon)\nu(x, y)}$$

and we are done. Otherwise, assume that $\text{diam}(f(A)) > \varepsilon \cdot \rho(f(x), f(y))$. Then there exists $f(a_1), f(a_2) \in f(A)$ for which $\rho(f(a_1), f(a_2)) > \varepsilon \cdot \rho(f(x), f(y))$, whereas

$$\nu(a_1, a_2) \leq \text{diam}(A) \leq \varepsilon \cdot \nu(A, B) \leq \varepsilon \cdot \nu(x, y).$$

Thus,

$$\frac{\rho(f(a_1), f(a_2))}{\nu(a_1, a_2)} > \frac{\varepsilon \cdot \rho(f(x), f(y))}{\varepsilon \cdot \nu(x, y)},$$

which is a contradiction to the maximality of the pair $\{x, y\}$. □

9. Fast approximation of the doubling dimension.

THEOREM 9.1. *Given a metric space \mathcal{M} with n points, one can approximate the doubling dimension dim of \mathcal{M} , up to a constant factor, in $2^{O(\text{dim})} n \log n$ expected time.*

Notice that this theorem, apart from its intrinsic interest, also removes the need to specify dim together with the input for the other algorithms in this paper.

The algorithm suggested in Theorem 9.1 naturally uses the net-tree.

PROPOSITION 9.2. *Given a net-tree T of a metric \mathcal{M} , denote by λ_T the maximum out degree in T . Then $\log \lambda_T$ is a constant approximation to $\text{dim}(\mathcal{M})$.*

Proof. Let $v \in T$ be the vertex with the maximum number of children λ_T . By Definition 2.1, any covering of $\mathbf{b}(\text{rep}_v, \frac{2\tau}{\tau-1}\tau^{\ell(v)})$ by balls of radius $\frac{\tau-5}{4\tau(\tau-1)}$ requires at least λ_T such balls. This means that $\text{dim}(\mathcal{M}) = \Omega(\log \lambda_T)$.

The upper bound $\dim(\mathcal{M}) = O(\log \lambda_T)$ follows easily from the arguments of section 7. There, we actually prove the existence of a $\lambda_T^{O(1)}$ -doubling measure in \mathcal{M} , and it is easy to prove that the existence of an α doubling measure in \mathcal{M} implies that $\dim(\mathcal{M}) \leq \alpha$. \square

Proof of Theorem 9.1. By Proposition 9.2 it is enough to show an implementation of the algorithm for constructing the net-tree that is oblivious to the doubling dimension of the metric. Checking the algorithm in section 3, we observe that the algorithms in sections 3.1, 3.3, and 3.4 are indeed oblivious to the doubling dimension. We are therefore left with describing a doubling-dimension-oblivious algorithm for constructing HST that $O(n^2)$ -approximates the given metric. More specifically, the only part that needs to be changed is the use of Lemma 2.4 in Lemma 3.5. To this end, instead of knowing λ , we “guess” the doubling constant to be 2^i , increasing i until we “succeed.” More accurately, in the i th iteration, we apply the following sampling step 2^{3i} times: Pick randomly a point p from P , and compute the ball $\mathbf{b}(p, r)$ of smallest radius around p containing at least $n/(2 \cdot 2^{3i})$ points. Next, consider the ball of radius $\mathbf{b}(p, 2r)$. If it contains $\leq n/2$ points, the algorithm succeeded, and it stops. The algorithm is guaranteed to stop when $i \geq \lceil \log n \rceil$. Denote by $\delta = \delta(X)$ the random value, which is the value of 2^i when the algorithm stopped, when applied to a point set $X \subset \mathcal{M}$.

The resulting spanner is a $3n$ -approximation *regardless* of the random bits, and thus the correctness of the net-tree algorithm is guaranteed. We need only argue about the expected running time for constructing the HST. The running time of the HST constructed is dominated by the spanner construction and the number of edges in it (see Lemma 3.5). Denote by λ the doubling constant of the metric \mathcal{M} .

PROPOSITION 9.3. *For any $X \subset \mathcal{M}$,*

1. $\mathbf{E}[\delta(X)^{-3}] \geq \lambda^{-3}/16$.
2. $\mathbf{E}[\delta(X)^3] = O(\lambda^3)$.

Proof. Consider the algorithm above for computing $\delta(X)$. Once i reaches the value $k = \lceil \log_2 \lambda \rceil$, the probability of success on each point sampled is at least 2^{-3k} (by the argument in Lemma 2.4). Hence the probability of success in the i th round, $i \geq k$, conditioned on a failure in all previous rounds is at least $1 - (1 - 2^{-3k})^{2^{3i}}$, which means that

$$\mathbf{E}[\delta(X)^{-3}] \geq 1 - (1 - 2^{-3k})^{2^{3k}} 2^{-3k} \geq (1 - 1/e)\lambda^{-3}/8.$$

It also means that $\Pr[\delta \geq 2^{k+i}] \leq (1 - 2^{-3k})^{2^{3(k+i-1)}} \leq \exp(-\binom{i}{2})$, and therefore

$$\mathbf{E}[\delta^3] = \sum_{t=1}^{\infty} \Pr[\delta^3 \geq t] \leq 2\lambda^3 + \sum_{t=2^{3k}}^{\infty} \exp\left(-\binom{\log t - 3k}{2}\right) \leq 2\lambda^3 + O(1). \quad \square$$

We prove only an upper bound on the running time. Bounding the number of edges is similar. Denote by $f(X)$ the running time of the algorithm when applied to $X \subseteq \mathcal{M}$, and let $g(X) = \mathbf{E}[f(X)]$ and $g(n) = \sup_{X \subseteq \mathcal{M}, |X|=n} g(X)$.

The spanner construction algorithm of Lemma 3.5 satisfies

$$(9.1) \quad g(X) \leq \mathbf{E}\left[\max_{\delta(X)^{-3} \leq \alpha \leq 1/2} (g(\alpha|X|) + g((1 - \alpha)|X|) + c'\delta(X)^3 n)\right].$$

We now prove by induction that $g(n) \leq c\lambda^3 n \ln n$ for some $c > 0$. Fix Y to be a

subset of \mathcal{M} of size n such that $g(Y) = g(n)$. We have

$$\begin{aligned}
 g(n) &\leq \mathbf{E} \left[\max_{\delta(Y)^{-3} \leq \alpha \leq 1/2} (\mathbf{E}[g(\alpha|Y)] + \mathbf{E}[g((1-\alpha)|Y)]) + c'\delta^3 n \right] \\
 &\leq \mathbf{E} \left[\max_{\delta(Y)^{-3} \leq \alpha \leq 1/2} (c\lambda^3 \alpha |Y| \ln(\alpha|Y|) + c\lambda^3 (1-\alpha) |Y| \ln((1-\alpha)|Y|) + c'\delta^3 n) \right] \\
 &\leq \mathbf{E} [c\lambda^3 \delta^{-3} n \ln(\delta^{-3} n) + c\lambda^3 (1-\delta^{-3}) n \ln((1-\delta^{-3}) n) + c'\delta^3 n] \\
 &\leq c\lambda^3 n \cdot \mathbf{E} [\delta^{-3} \ln(\delta^{-3} n) + (1-\delta^{-3}) \ln((1-\delta^{-3}) n)] + (c''\lambda^3 + d')n \\
 &\leq c\lambda^3 n \cdot \mathbf{E} [\ln((1-\delta^{-3}) n)] + (c''\lambda^3 + d')n \\
 &\leq c\lambda^3 n \ln n + c\lambda^3 n \cdot \mathbf{E} [\ln(1-\delta^{-3})] + (c''\lambda^3 + d')n \\
 &\leq c\lambda^3 n \ln n - c\lambda^3 n \cdot \mathbf{E} [\delta^{-3}] + (c''\lambda^3 + d')n \\
 &\leq c\lambda^3 n \ln n - c\lambda^3 n \cdot (\lambda^{-3}/16) + (c''\lambda^3 + d')n \\
 &\leq c\lambda^3 n \ln n,
 \end{aligned}$$

since $\ln(1-\delta^{-3}) \leq -\delta^{-3}$, by Proposition 9.3, and for $c > 0$ large enough. \square

10. Concluding remarks. In this paper, we show how to efficiently construct hierarchical nets for finite spaces with low doubling dimension and use this construction in several applications. We believe that this result will have further applications.

Among other things, our fast construction of WSPD implies a near linear time construction of an approximate minimum spanning tree of the space. Our fast construction of net-tree implies that one can do 2-approximate k -center clustering in $O(n \log n)$ expected time.

Further, transfer of problems and techniques from low-dimensional Euclidean space to low-dimensional metrics seems to be interesting. A plausible example of such problems is the construction of $(1 + \varepsilon)$ -spanners with some additional properties (such as low total weight or small hop-diameter). Results of this flavor exist in low-dimensional Euclidean spaces.

It is easy to verify that, for a general metric, no HST can be constructed without inspecting all $\binom{n}{2}$ edges. Indeed, consider the uniform metric over n points, and change in an adversarial fashion a single edge to have length 0.

10.1. All nearest neighbors. The *all nearest neighbor problem* is to compute for a set P of n points the (exact) nearest neighbor for each point of $p \in P$ in the set $P \setminus \{p\}$. It is known that in low-dimensional Euclidean space this can be done in $O(n \log n)$ time [13, 46, 11]. One can ask if a similar result can be attained for finite metric spaces with low doubling dimensions. Below we show that this is impossible.

Consider the points p_1, \dots, p_n , where the distance between p_i and p_j , for $i < j$, is either 2^j or $2^j + \varepsilon$, for $\varepsilon < 0.1$. It is easy to verify that this metric has doubling constant at most three. We now show that for any deterministic algorithm for computing all nearest neighbors, there is a metric in the family of the metrics described above for which the algorithm performs $\binom{n}{2}$ distance queries.

This claim is proved using an adversarial argument: When the adversary is queried about the distance between p_i and p_j , for $i < j$, if not all the distances between p_1, \dots, p_{j-1} and p_j were specified, the adversary will always return the distance to be $2^j + \varepsilon$. The distances 2^j would be returned only for the last pair among the $j - 1$ pairs in this set. In particular, for the algorithm to know what is the closest point to p_j , it must perform $j - 1$ queries. Thus, overall, an algorithm doing all nearest neighbors for p_1, \dots, p_n will have to perform $\binom{n}{2}$ queries.

A similar asymptotic lower bound can be proved for randomized algorithms using Yao’s principle (here the adversary selects for each j one index $i_j < j$ at random for which $d(p_{i_j}, p_j) = 2^j$, and for the rest of $i \neq i_j, i < j, d(p_i, j) = 2^j + \varepsilon$).

At this point, it is natural to ask whether one can achieve running time of $O(n \log(n\Phi(P)))$ for the all nearest neighbor problem. This, however, is straightforward. Indeed, compute a 4-WSPD of P . Clearly, if q is a nearest neighbor for p , then there is a pair in the WSPD such that p is the only point on one side, and the other side contains q . Thus, we scan all such unbalanced pairs (one point on one side, and many points on the other side) and compute the nearest neighbor for each point. Thus, this computes all nearest neighbors. As for the running time analysis, consider all such pairs in distance range l to $2l$, and observe that by a packing argument, for any node u in the net-tree, the number of such WSPD pairs with u in them is $2^{O(\dim)}$. In fact, along a path in the net-tree, only a constant number of nodes might participate in such pairs. Thus, every point is being scanned $2^{O(\dim)}$ times, implying that scanning all such pairs takes $2^{O(\dim)}n$ time. There are $\lceil \lg(\Phi(P)) \rceil$ resolutions, so the overall running time is $2^{O(\dim)}n \log(n\Phi(P))$.

Appendix. Proof of Lemma 3.9. Notice that Lemma 3.7 implies that $(\bar{r}_i)_{i \geq 1}$ is a monotone nonincreasing sequence and that $r_i \geq \bar{r}_i \geq r_i/(1 + n^{-2}) \geq \frac{4}{5}r_i$.

Proof of Lemma 3.9. We prove by induction on k all five assertions together. The base case is obvious. Assume by the induction hypothesis that $T^{(k-1)}$ satisfies all the properties above. We prove it for $T^{(k)}$.

Property (i). Every point inserted during the l th phase (i.e., a point p_i for which $\lceil \log_\tau \bar{r}_i \rceil = l$) must have its current parent (in $T^{(k)}$) at level l . Thus, if $\ell(\hat{u}) > l$, this means that c_{p_k} was inserted before the current phase, which means that it is indeed the closest point to p_k among $\{p_1, \dots, p_h\}$. Otherwise, if $\ell(\hat{u}) = l$, then

$$\begin{aligned} d_{\mathcal{M}}(\hat{u}, q) &\leq d_{\mathcal{M}}(\hat{u}, c_{p_k}) + d_{\mathcal{M}}(c_{p_k}, p_k) + d_{\mathcal{M}}(p_k, q) \\ &\leq 2 \cdot \tau^l + (1 + n^{-2})\tau^l + (1 + n^{-2})\tau^l \leq 13 \cdot \tau^l. \end{aligned}$$

Since q appears before the level l began, either $\ell(\bar{p}(q)) > l$ and then $q \in \overline{\text{Rel}}(\hat{u})$, or $\ell(\bar{p}(q)) = l$; but then it must be that $\text{rep}_{\bar{p}(q)} = q$, so $\bar{p}(q) \in \overline{\text{Rel}}(\hat{u})$. Either case q is a representative of a vertex in $\overline{\text{Rel}}(\hat{u})$ which is the same as $\text{Rel}(\hat{u})$ (in $T^{(k-1)}$).

Property (ii). We shall prove it for both p_k and the new internal vertex (in case (a) of the construction). Consider first case (a) of the construction:

$$d_{\mathcal{M}}(\text{rep}_u, \text{rep}_v) = d_{\mathcal{M}}(\text{rep}_u, q) \leq \tau^{\ell(u)},$$

where the last inequality follows from the induction hypothesis. Also,

$$d_{\mathcal{M}}(\text{rep}_v, p_k) \leq 2 \cdot \tau^{\ell(v)},$$

and we are done with the first case of the construction.

Case (b) of the construction follows from the definition of q and since, as argued above, for $u = \bar{p}(q)$, $\text{rep}_u = q$.

Property (iii). Fix some $t \in \mathfrak{R}$, and let x and y be two vertices for which $\max\{\ell(x), \ell(y)\} < t \leq \min\{\ell(\bar{p}(x)), \ell(\bar{p}(y))\}$. If both x and y are not p_k , then the claim follows from the inductive hypothesis (even for the newly formed internal vertex, since it inherits its parent and representative from a previously established vertex). Otherwise, assume $x = p_k$. As p_k is the latest addition of a leaf to T , $d_{\mathcal{M}}(p_k, \text{rep}_y) \geq \bar{r}_{k-1} \geq \tau^{t-1}$. Note that $\ell(\bar{p}(p_k)) = l$, so $t \leq l$, and we conclude that $d_{\mathcal{M}}(\text{rep}_x, \text{rep}_y) \geq \tau^{t-1}$.

Property (iv). We next prove that $T^{(k)}$ is a net-tree. The only nonstraightforward claims are the packing and covering properties. The covering property follows from Property (ii) of this lemma: Let $u = u_1$ be a vertex, $v = u_m$ a descendant, and $\langle u_1, \dots, u_m \rangle$ the path between them in T ; then

$$\begin{aligned} d_{\mathcal{M}}(\text{rep}_u, \text{rep}_v) &\leq \sum_{i=1}^{m-1} d_{\mathcal{M}}(\text{rep}_{u_i}, \text{rep}_{u_{i+1}}) \leq 2 \sum_{i=1}^{m-1} \tau^{\ell(u_i)} \\ &\leq 2 \sum_{i=1}^{m-1} \tau^{\ell(u_1) - (i-1)} \leq \frac{2\tau}{\tau - 1} \cdot \tau^{\ell(u)}. \end{aligned}$$

The packing property is more delicate. Let w be an arbitrary vertex in $T^{(k)}$, and let $x \notin P_w$ be a point. We want to prove that $d_{\mathcal{M}}(x, \text{rep}_w) \geq \frac{\tau-5}{2(\tau-1)} \tau^{\ell(\bar{p}(w))-1}$. Let $\hat{x} \in T^{(k)}$ be an ancestor of x such that $\ell(\hat{x}) \leq \ell(\bar{p}(w)) - 1 < \ell(\bar{p}(\hat{x}))$. Applying Property (iii) with $t = \ell(\bar{p}(w))$, we get that $d_{\mathcal{M}}(\text{rep}_{\hat{x}}, \text{rep}_w) \geq \tau^{\ell(\bar{p}(w))-1}$.

If $x = \hat{x}$, we are done. Otherwise, if $\ell(\hat{x}) < \ell(\bar{p}(w)) - 1$, then, by Property (ii), we have

$$\begin{aligned} d_{\mathcal{M}}(\text{rep}_w, x) &\geq d_{\mathcal{M}}(\text{rep}_w, \text{rep}_{\hat{x}}) - d_{\mathcal{M}}(\text{rep}_{\hat{x}}, x) \\ &\geq \tau^{\ell(\bar{p}(w))-1} - \frac{2\tau}{\tau - 1} \cdot \tau^{\ell(\bar{p}(w))-2} = \frac{\tau - 3}{\tau - 1} \cdot \tau^{\ell(\bar{p}(w))-1}, \end{aligned}$$

and we are done.

Otherwise, let $\bar{x} \in T^{(k)}$ be an ancestor of x which is the child of \hat{x} ($\bar{p}(\bar{x}) = \hat{x}$). If $\text{rep}_{\bar{x}} = \text{rep}_{\hat{x}}$, then the preceding argument (where $\ell(\hat{x}) < \ell(\bar{p}(w)) - 1$) also applies here, and we are done.

Otherwise, we get the following situation: $\ell(\bar{p}(\bar{x})) = \ell(\bar{p}(w)) - 1$ and $\text{rep}_{\bar{x}} \neq \text{rep}_{\bar{p}(\bar{x})}$. But this can happen only if \bar{x} was inserted during level $\ell(\bar{p}(w)) - 1$. Recall that the algorithm connects $\text{rep}_{\bar{x}}$ as a child of a vertex in level $\ell(\bar{p}(w)) - 1$ whose representative is the closest point among those appearing during the levels greater than $\ell(\bar{p}(w)) - 1$. Note that both $\text{rep}_{\bar{p}(\bar{x})}$ and rep_w are inserted in a level greater than $\ell(\bar{p}(w)) - 1$. We conclude that $d_{\mathcal{M}}(\text{rep}_{\bar{x}}, \text{rep}_{\bar{p}(\bar{x})}) \leq d_{\mathcal{M}}(\text{rep}_{\bar{x}}, \text{rep}_w)$. Therefore

$$\begin{aligned} d_{\mathcal{M}}(\text{rep}_{\bar{x}}, \text{rep}_w) &\geq \max \left\{ d_{\mathcal{M}}(\text{rep}_{\bar{x}}, \text{rep}_{\bar{p}(\bar{x})}), d_{\mathcal{M}}(\text{rep}_w, \text{rep}_{\bar{p}(\bar{x})}) - d_{\mathcal{M}}(\text{rep}_{\bar{x}}, \text{rep}_{\bar{p}(\bar{x})}) \right\} \\ &\geq \frac{d_{\mathcal{M}}(\text{rep}_w, \text{rep}_{\bar{p}(\bar{x})})}{2} \geq 0.5 \cdot \tau^{\ell(\bar{p}(w))-1}. \end{aligned}$$

Hence, by the covering property,

$$\begin{aligned} d_{\mathcal{M}}(x, \text{rep}_w) &\geq d_{\mathcal{M}}(\text{rep}_{\bar{x}}, \text{rep}_w) - d_{\mathcal{M}}(\text{rep}_{\bar{x}}, x) \geq 0.5 \cdot \tau^{\ell(\bar{p}(w))-1} - \frac{2\tau}{\tau - 1} \cdot \tau^{\ell(\bar{p}(w))-2} \\ &= \frac{\tau - 5}{2(\tau - 1)} \cdot \tau^{\ell(\bar{p}(w))-1}, \end{aligned}$$

and we are done.

Property (v). Assume that a new vertex x is attached as a child to a vertex y . We shall prove that our traversing algorithm visits all vertices w for which either $w \in \overline{\text{Rel}}(x)$ or $x \in \overline{\text{Rel}}(w)$. Suppose first that $x \in \overline{\text{Rel}}(w)$. Thus, $\ell(w) < \ell(y)$. Let z be an ancestor of w for which $\ell(z) \leq \ell(y) < \ell(\bar{p}(z))$. Let $\langle z = z_1, \dots, z_m = w \rangle$ be the

path between them in T . Then, for any $1 \leq i \leq m-1$, it holds that

$$\begin{aligned} d_{\mathcal{M}}(\text{rep}_x, \text{rep}_{z_i}) &\leq d_{\mathcal{M}}(\text{rep}_x, \text{rep}_w) + d_{\mathcal{M}}(\text{rep}_{z_i}, \text{rep}_w) \\ &\leq 13 \cdot \tau^{\ell(z_m)} + \frac{2\tau}{\tau-1} \cdot \tau^{\ell(z_i)} \leq 13 \cdot \tau^{\ell(z_i)}. \end{aligned}$$

Thus $x \in \overline{\text{Rel}}(z_i)$ for any $2 \leq i \leq m$. Thus, if $z = z_1 \in \text{Rel}(y)$, we are assured that $w = z_m$ will be visited. Indeed, $z \in \overline{\text{Rel}}(y)$, since

$$\begin{aligned} d_{\mathcal{M}}(\text{rep}_y, \text{rep}_z) &\leq d_{\mathcal{M}}(\text{rep}_y, \text{rep}_x) + d_{\mathcal{M}}(\text{rep}_x, \text{rep}_w) + d_{\mathcal{M}}(\text{rep}_w, \text{rep}_z) \\ &\leq 2 \cdot \tau^{\ell(y)} + 13 \cdot \tau^{\ell(w)} + \frac{2\tau}{\tau-1} \cdot \tau^{\ell(z)} \\ &\leq 2 \cdot \tau^{\ell(y)} + 13 \cdot \tau^{\ell(y)-1} + \frac{2\tau}{\tau-1} \cdot \tau^{\ell(y)} \\ &\leq 13 \cdot \tau^{\ell(y)}. \end{aligned}$$

This means that $z \in \text{Rel}(y)$ by the inductive hypothesis.

Next, we consider the case when $w \in \overline{\text{Rel}}(x)$. In this case, $\ell(w) \leq \ell(x) < \ell(\overline{\text{p}}(w))$ and

$$d_{\mathcal{M}}(\text{rep}_w, \text{rep}_y) \leq d_{\mathcal{M}}(\text{rep}_w, \text{rep}_x) + d_{\mathcal{M}}(\text{rep}_x, \text{rep}_y) \leq 13 \cdot \tau^{\ell(x)} + 2 \cdot \tau^{\ell(y)} \leq 13 \cdot \tau^{\ell(y)}.$$

Hence, if $\ell(\overline{\text{p}}(w)) > \ell(y)$, then $w \in \overline{\text{Rel}}(y)$ which implies that $w \in \text{Rel}(y)$ by the inductive hypothesis, and we are done.

If $\ell(\overline{\text{p}}(w)) = \ell(y)$, then

$$\begin{aligned} d_{\mathcal{M}}(\text{rep}_{\overline{\text{p}}(w)}, \text{rep}_y) &\leq d_{\mathcal{M}}(\text{rep}_{\overline{\text{p}}(w)}, \text{rep}_w) + d_{\mathcal{M}}(\text{rep}_w, \text{rep}_x) + d_{\mathcal{M}}(\text{rep}_x, \text{rep}_y) \\ &\leq 2 \cdot \tau^{\ell(y)} + 13 \cdot \tau^{\ell(y)-1} + 2 \cdot \tau^{\ell(y)} \leq 13 \cdot \tau^{\ell(y)}. \end{aligned}$$

Thus, in this case $\overline{\text{p}}(w) \in \overline{\text{Rel}}(y)$, and using the inductive hypothesis, we are done.

We are left with the case $\ell(\overline{\text{p}}(w)) < \ell(y)$. In this case

$$\begin{aligned} d_{\mathcal{M}}(\text{rep}_{\overline{\text{p}}(w)}, \text{rep}_x) &\leq d_{\mathcal{M}}(\text{rep}_{\overline{\text{p}}(w)}, \text{rep}_w) + d_{\mathcal{M}}(\text{rep}_w, \text{rep}_x) \\ &\leq 2 \cdot \tau^{\ell(\overline{\text{p}}(w))} + 13 \cdot \tau^{\ell(\overline{\text{p}}(w))-1} \leq 13 \cdot \tau^{\ell(\overline{\text{p}}(w))}. \end{aligned}$$

Thus, we have that $x \in \text{Rel}(\overline{\text{p}}(w))$. As was proved above, this means that $\overline{\text{p}}(w)$ will be visited, and since x is added to $\text{Rel}(\overline{\text{p}}(w))$, the algorithm also visits the children of $\overline{\text{p}}(w)$, and in particular, w . \square

Acknowledgments. We thank James Lee, Ken Clarkson, and Alex Slivkins for helpful correspondence regarding the content of this paper. James Lee pointed us to some recent related papers and suggested the construction of a doubling measure as an application of the net-tree. The observations about the all nearest neighbors problem rose from discussions with Ken Clarkson. Alex Slivkins pointed out to us an error in a preliminary version of this paper. Finally, the authors would like to thank the anonymous referees for their useful comments.

REFERENCES

- [1] S. ARYA, D. M. MOUNT, N. S. NETANYAHU, R. SILVERMAN, AND A. Y. WU, *An optimal algorithm for approximate nearest neighbor searching in fixed dimensions*, J. ACM, 45 (1998), pp. 891–923.
- [2] P. ASSOUD, *Plongements lipschitziens dans \mathbf{R}^n* , Bull. Soc. Math. France, 111 (1983), pp. 429–448.
- [3] A. BARVINOK, *A Course in Convexity*, Grad. Stud. Math. 54, AMS, Providence, RI, 2002.
- [4] M. A. BENDER AND M. FARACH-COLTON, *The lca problem revisited*, in Proceedings of the 4th Latin American Symposium on Theoretical Informatics, Springer-Verlag, Berlin, 2000, pp. 88–94.
- [5] M. A. BENDER AND M. FARACH-COLTON, *The level ancestor problem simplified*, Theoret. Comput. Sci., 321 (2004), pp. 5–12.
- [6] J. L. BENTLEY AND J. B. SAXE, *Decomposable searching problems i: Static-to-dynamic transformation*, J. Algorithms, 1 (1980), pp. 301–358.
- [7] Y. BENYAMINI AND J. LINDENSTRAUSS, *Geometric Nonlinear Functional Analysis*, AMS, Providence, RI, 2000.
- [8] A. BEYGELZIMER, S. KAKADE, AND J. LANGFORD, *Cover Trees for Nearest Neighbor*, http://ttic.uchicago.edu/~sham/papers/ml/cover_tree.pdf (2004).
- [9] M. BÄDOIU, K. DHAMDHARE, A. GUPTA, Y. RABINOVICH, H. RÄCKE, R. RAVI, AND A. SIDIROPOULOS, *Approximation algorithms for low-distortion embedding into low-dimensional spaces*, in Proceedings of the 16th Annual ACM–SIAM Symposium on Discrete Algorithms, SIAM, Philadelphia, ACM, New York, 2005, pp. 119–128.
- [10] P. B. CALLAHAN, *The Well Separated Pair Decomposition and Its Application*, Tech. report, Johns Hopkins University, Baltimore, MD, 1995.
- [11] P. B. CALLAHAN AND S. R. KOSARAJU, *A decomposition of multidimensional point sets with applications to k -nearest-neighbors and n -body potential fields*, J. Assoc. Comput. Mach., 42 (1995), pp. 67–90.
- [12] H. T.-H. CHAN, A. GUPTA, B. M. MAGGS, AND S. ZHOU, *On hierarchical routing in doubling metrics*, in Proceedings of the 16th Annual ACM–SIAM Symposium on Discrete Algorithms, SIAM, Philadelphia, ACM, New York, 2005, pp. 762–771.
- [13] K. L. CLARKSON, *Fast algorithms for the all nearest neighbors problem*, in Proceedings of the 24th Annual IEEE Symposium on Foundations of Computer Science, IEEE, Los Alamitos, CA, 1983, pp. 226–232.
- [14] K. L. CLARKSON, *Nearest neighbor queries in metric spaces*, Discrete Comput. Geom., 22 (1999), pp. 63–93.
- [15] K. L. CLARKSON, *Nearest Neighbor Searching in Metric Spaces: Experimental Results for $sb(s)$* , http://cm.bell-labs.com/who/clarkson/Msb/white_paper.pdf (2002).
- [16] K. L. CLARKSON AND P. W. SHOR, *Applications of random sampling in computational geometry*, II, Discrete Comput. Geom., 4 (1989), pp. 387–421.
- [17] T. FEDER AND D. H. GREENE, *Optimal algorithms for approximate clustering*, in Proceedings of the 20th Annual ACM Symposium on Theory of Computing, ACM, New York, 1988, pp. 434–444.
- [18] C. GAVOILLE, M. KATZ, N. A. KATZ, C. PAUL, AND D. PELEG, *Approximate distance labeling schemes*, in Proceedings of the 9th Annual European Symposium on Algorithms, Lecture Notes in Comput. Sci. 2161, Springer, Berlin, 2001, pp. 476–487.
- [19] C. GAVOILLE, D. PELEG, S. PERENNES, AND R. RAZ, *Distance labeling in graphs*, J. Algorithms, 53 (2004), pp. 85–112.
- [20] T. GONZALEZ, *Clustering to minimize the maximum intercluster distance*, Theoret. Comput. Sci., 38 (1985), pp. 293–306.
- [21] J. GUDMUNDSSON, C. LEVCOPOULOS, G. NARASIMHAN, AND M. SMID, *Approximate distance oracles for geometric graphs*, in Proceedings of the 13th Annual ACM–SIAM Symposium on Discrete Algorithms, SIAM, Philadelphia, ACM, New York, 2002, pp. 828–837.
- [22] J. GUDMUNDSSON, C. LEVCOPOULOS, G. NARASIMHAN, AND M. H. M. SMID, *Approximate distance oracles revisited*, in Proceedings of the 13th Annual International Symposium on Algorithms and Computation, Springer, Berlin, 2002, pp. 357–368.
- [23] A. GUPTA, R. KRAUTHGAMER, AND J. R. LEE, *Bounded geometries, fractals, and low-distortion embeddings*, in Proceedings of the 44th Annual IEEE Symposium on Foundations of Computer Science, IEEE, Los Alamitos, CA, 2003, pp. 534–543.
- [24] S. HAR-PELED, *A replacement for Voronoi diagrams of near linear size*, in Proceedings of the 42nd Annual IEEE Symposium on Foundations of Computer Science, IEEE, Los Alamitos, CA, 2001, pp. 94–103.

- [25] S. HAR-PELED, *Clustering motion*, Discrete Comput. Geom., 31 (2004), pp. 545–565.
- [26] S. HAR-PELED AND S. MAZUMDAR, *Coresets for k -means and k -median clustering and their applications*, in Proceedings of the 36th Annual ACM Symposium on Theory of Computing, ACM, New York, 2004, pp. 291–300.
- [27] J. HEINONEN, *Lectures on Analysis on Metric Spaces*, Universitext, Springer-Verlag, New York, 2001.
- [28] K. HILDRUM, J. KUBIATOWICZ, S. MA, AND S. RAO, *A note on the nearest neighbor in growth-restricted metrics*, in Proceedings of the 15th Annual ACM–SIAM Symposium on Discrete Algorithms, SIAM, Philadelphia, ACM, New York, 2004, pp. 553–554.
- [29] P. INDYK AND R. MOTWANI, *Approximate nearest neighbors: Towards removing the curse of dimensionality*, in Proceedings of the 30th Annual ACM Symposium on Theory of Computing, ACM, New York, 1998, pp. 604–613.
- [30] D. R. KARGER AND M. RUHL, *Finding nearest neighbors in growth-restricted metrics*, in Proceedings of the 34th Annual ACM Symposium on Theory of Computing, ACM, New York, 2002, pp. 741–750.
- [31] J. M. KEIL, *Approximating the complete Euclidean graph*, in Proceedings of the 1st Scandinavian Workshop on Algorithm Theory, Lecture Notes in Comput. Sci. 318, Springer, Berlin, 1988, pp. 208–213.
- [32] D. E. KNUTH, *The Art of Computer Programming: Seminumerical Algorithms*, 3rd ed., Addison-Wesley, New York, 1997.
- [33] R. KRAUTHGAMER AND J. R. LEE, *The black-box complexity of nearest neighbor search*, in Proceedings of the 31st International Colloquium on Automata, Languages and Programming, Springer, Berlin, 2004, pp. 858–869.
- [34] R. KRAUTHGAMER AND J. R. LEE, *Navigating nets: Simple algorithms for proximity search*, in Proceedings of the 15th Annual ACM–SIAM Symposium on Discrete Algorithms, SIAM, Philadelphia, ACM, New York, 2004, pp. 791–800.
- [35] T. J. LAAKSO, *Plane with A_∞ -weighted metric not bi-Lipschitz embeddable to \mathbb{R}^N* , Bull. London Math. Soc., 34 (2002), pp. 667–676.
- [36] J. LUUKKAINEN AND E. SAKSMAN, *Every complete doubling metric space carries a doubling measure*, Proc. Amer. Math. Soc., 126 (1998), pp. 531–534.
- [37] J. MATOUSEK, *Using the Borsuk-Ulam Theorem. Lectures on Topological Methods in Combinatorics and Geometry*, Universitext, Springer-Verlag, Berlin, 2003.
- [38] M. H. OVERMARS AND J. VAN LEEUWEN, *Maintenance of configurations in the plane*, J. Comput. System Sci., 23 (1981), pp. 166–204.
- [39] D. PELEG, *Informative labeling schemes for graphs*, Theoret. Comput. Sci., 340 (2005), pp. 577–593.
- [40] D. PELEG AND A. SCHÄFFER, *Graph spanners*, J. Graph Theory, 13 (1989), pp. 99–116.
- [41] S. SEMMES, *On the nonexistence of bi-Lipschitz parameterizations and geometric problems about A_∞ -weights*, Rev. Mat. Iberoamericana, 12 (1996), pp. 337–410.
- [42] A. SLIVKINS, *Distance Estimation and Object Location via Rings of Neighbors*, Tech. Report TR2005-1977, Cornell CIS, Cornell University, Ithaca, NY, 2005.
- [43] A. SLIVKINS, *Distributed approaches to triangulation and embedding*, in Proceedings of the 16th Annual ACM–SIAM Symposium on Discrete Algorithms, SIAM, Philadelphia, ACM, New York, 2005, pp. 640–649.
- [44] K. TALWAR, *Bypassing the embedding: Algorithms for low dimensional metrics*, in Proceedings of the 36th Annual ACM Symposium on Theory of Computing, ACM, New York, 2004, pp. 281–290.
- [45] M. THORUP AND U. ZWICK, *Approximate distance oracles*, in Proceedings of the 33rd Annual ACM Symposium on Theory of Computing, ACM, New York, 2001, pp. 183–192.
- [46] P. M. VAIDYA, *An $o(n \log n)$ algorithm for the all-nearest-neighbors problem*, Discrete Comput. Geom., 4 (1989), pp. 101–115.
- [47] A. L. VOL'BERG AND S. V. KONYAGIN, *On measures with the doubling condition*, Izv. Akad. Nauk SSSR Ser. Mat., 51 (1987), pp. 666–675.
- [48] J.-M. WU, *Hausdorff dimension and doubling measures on metric spaces*, Proc. Amer. Math. Soc., 126 (1998), pp. 1453–1459.

EXTRACTING RANDOMNESS VIA REPEATED CONDENSING*

OMER REINGOLD[†], RONEN SHALTIEL[‡], AND AVI WIGDERSON[§]

Abstract. Extractors (as defined by Nisan and Zuckerman) are procedures that use a *small* number of truly random bits (called the seed) to extract *many* (almost) truly random bits from arbitrary distributions as long as distributions have sufficient (min)-entropy. A natural weakening of an extractor is a *condenser*, whose output distribution has a higher entropy rate than the input distribution (without losing much of the initial entropy). An extractor can be viewed as an ultimate condenser because it outputs a distribution with the maximal entropy rate.

In this paper we construct explicit condensers with short seed length. The condenser constructions combine (variants of or more efficient versions of) ideas from several works, including the block extraction scheme of [N. Nisan and D. Zuckerman, *J. Comput. System Sci.*, 52 (1996), pp. 43–52], the observation made in [A. Srinivasan and D. Zuckerman, *SIAM J. Comput.*, 28 (1999), pp. 1433–1459; N. Nisan and A. Ta-Shma, *J. Comput. System Sci.*, 58 (1999), pp. 148–173] that a failure of the block extraction scheme is also useful, the recursive “win-win” case analysis of [R. Impagliazzo, R. Shaltiel, and A. Wigderson, *Near-optimal conversion of hardness into pseudo-randomness*, in Proceedings of the 40th Annual IEEE Symposium on Foundations of Computer Science, IEEE, Los Alamitos, CA, 1999, pp. 181–190; R. Impagliazzo, R. Shaltiel, and A. Wigderson, *Extractors and pseudo-random generators with optimal seed length*, in Proceedings of the 32nd Annual ACM Symposium on Theory of Computing, ACM, New York, 2000, pp. 1–10], and the error correction of random sources used in [L. Trevisan, *J. ACM*, 48 (2001), pp. 860–879].

As a by-product (via repeated iterating of condensers), we obtain new extractor constructions. The new extractors give significant qualitative improvements over previous ones for sources of arbitrary min-entropy; they are nearly optimal *simultaneously* in the two main parameters of seed length and output length. Specifically, our extractors can make any one of these two parameters optimal (up to a constant factor) only at a *polylogarithmic* loss in the other. Previous constructions require *polynomial* loss in both cases for general sources.

We also give a simple reduction converting “standard” extractors (which are good for an average seed) into “strong” ones (which are good for most seeds), with essentially the same parameters. With this reduction, all the above improvements apply to strong extractors as well.

Key words. randomness extractors, randomness condensers, derandomization

AMS subject classifications. 68Q99, 68R05

DOI. 10.1137/S0097539703431032

1. Introduction.

1.1. Extractors. One of the most successful ideas in modern computer science is that of probabilistic algorithms and protocols. These are procedures that use random coin tosses when performing computational tasks. A natural problem in these procedures is how computers can obtain truly random bits.

*Received by the editors July 6, 2003; accepted for publication (in revised form) September 30, 2005; published electronically March 3, 2006. A preliminary version of this paper appeared in [18].
<http://www.siam.org/journals/sicomp/35-5/43103.html>

[†]Incumbent of the Walter and Elise Haas Career Development Chair, Department of Computer Science, Weizmann Institute of Science, Rehovot, Israel (omer.reingold@weizmann.ac.il). Most of this author’s research was performed while at AT&T Labs–Research, Florham Park, NJ, and while visiting the Institute for Advanced Study, Princeton, NJ. This author’s research was supported in part by United States–Israel Binational Science Foundation grant 2002246.

[‡]Department of Computer Science, University of Haifa, Haifa 31905, Israel (ronen@cs.haifa.ac.il). Part of this author’s research was performed while visiting the Institute for Advanced Study, Princeton, NJ. This author’s research was supported in part by the Borland Scholar program.

[§]Institute for Advanced Study, Princeton, NJ 08540 (avi@ias.edu). This author’s research was supported by United States–Israel Binational Science Foundation grant 97-00188.

A line of research initiated by [1, 20, 28] was motivated by the question of availability of truly random bits. The idea behind this question is to make truly random bits available by refining the (imperfect) randomness found in some natural physical processes, with the ultimate goal of designing procedures called “randomness extractors” that, given a sample from an arbitrary source of randomness, produce truly random bits. It was shown by [20] that this goal cannot be achieved by deterministic algorithms, even for some randomness sources that have a simple and “nice” structure. In light of this, the goal became to “spend” as few as possible truly random bits in order to extract as many as possible (almost) truly random bits from arbitrary imperfect random sources which contain sufficient randomness.

The most general definition of weak random sources and the formal definition of extractors emerged from the works of [3, 30, 31, 13]. The definition of extractors [13] requires quantifying two notions: the first is the amount of randomness in probability distributions, which is measured using a variant of the entropy function called *min-entropy*.

DEFINITION 1.1. *A distribution X is called a k -source if the probability it assigns to every element in its range is bounded above by 2^{-k} . The min-entropy of X (denoted by $H_\infty(X)$) is the maximal k such that X is a k -source.*

The second notion is the quality of the extracted bits, which is measured using the statistical distance between the extracted bits and truly uniform ones.

DEFINITION 1.2. *Two distributions P, Q (over the same domain Ω) are ϵ -close if they have a statistical distance of at most ϵ . (For every event $A \subseteq \Omega$, the probability that both distributions assign to A differs by at most ϵ).*

Extractors are functions that use *few* truly random bits to extract *many* (almost) truly random bits from arbitrary distributions which “contain” sufficient randomness. A formal definition follows. We use U_m to denote the uniform distribution on m bit strings.

DEFINITION 1.3 (see [13]). *A function $Ext : \{0, 1\}^n \times \{0, 1\}^d \rightarrow \{0, 1\}^m$ is a (k, ϵ) -extractor if for every k -source X the distribution $Ext(X, U_d)$ (obtained by running the extractor on an element sampled from X and a uniformly chosen d bit string, which we call the seed) is ϵ -close to U_m . The entropy-loss of an extractor is defined to be $k + d - m$.*

Informally, we will say that an extractor uses a seed of length d in order to extract m bits from distributions on n bits which contain k random bits. We refer to the ratio between m and k as the fraction of the randomness which the extractor extracts, and to the ratio between k and n as the entropy rate of the source.

Apart from their original application of obtaining random bits from natural sources, extractors turned out to be useful in many areas in complexity theory and combinatorics, with examples being construction of pseudorandom generators for space bounded computation; deterministic amplification; oblivious sampling; constructive leader election; and explicit constructions of expander graphs, superconcentrators, and sorting networks. The reader is referred to the excellent survey articles [11, 12]. (A more recent survey article that complements the aforementioned ones is [21].)

1.2. Extractor constructions: Goals and previous work. We now survey some of the goals of extractor constructions and the previous research done towards achieving these goals. Extractor constructions are measured by viewing d and m as functions of the source parameters (n and k) and the required error ϵ . A recent result of [16] enables us to rid ourselves of ϵ and concentrate on the case that ϵ is

TABLE 1

Extracting a constant fraction: $m = (1 - \alpha)k$ for arbitrary $\alpha > 0$.

Reference	min-entropy k	Seed length d
[32]	$k = \Omega(n)$	$O(\log n)$
[24]	any k	$O(\log^9 n)$
[7]	$k = 2^{O(\sqrt{\log n})}$	$O(\log n \cdot \log \log \log n)$
[17]	any k	$O(\log^2 n)$
Thm. 1.4	any k	$O(\log n \cdot (\log \log n)^2)$
optimal	any k	$O(\log n)$

some fixed small constant.¹ We maintain this convention throughout the introduction.

When constructing extractors, there are two possible objectives: minimizing the seed length d and maximizing the output size m . It should be noted that the existence of an optimal extractor (which optimizes both parameters simultaneously, and matches the known lower bounds due to [14]) can be easily proven using the probabilistic method. Thus, the goal is to match this performance with explicit constructions. A (family of) extractors is *explicit* if it can be computed in polynomial time.²

In the remainder of this subsection we survey the currently known explicit extractors constructions known for the two objectives. (The reader is also referred to [21] for a more recent survey that covers some subsequent work [25, 26, 22, 9].) Tables 1 and 2 contain some extractor constructions but are far from complete in their coverage of the mass of work done in this area. In the following paragraphs we focus on extractors which work for arbitrary min-entropy threshold k .

1. *Minimizing the seed length.* A lower bound of $\Omega(\log n)$ on the seed length was given in [13, 14]. In contrast to the (nonexplicit) optimal extractor, which uses a seed of length $O(\log n)$ to extract *all* the randomness from the source, explicit constructions of extractors can optimize the seed length only at the cost of extracting a small fraction of the randomness. For large k ($k = \Omega(n)$) the extractor of [32] uses a seed of length $O(\log n)$ to extract a constant fraction of the initial randomness. However, for smaller k , explicit constructions can extract only a polynomial fraction of the initial randomness ($m = k^{1-\alpha}$ for an arbitrary constant α). This result was first achieved in [27] for $k = n^{\Omega(1)}$ and later extended for any k in [7].

2. *Maximizing the output size.* Current constructions of explicit extractors can extract large fractions of the randomness only at the cost of enlarging the seed length. A general method for increasing the fraction extracted at the cost of enlarging the seed length was given in [29]. The best extractors that extract *all* the randomness out of random sources are constructed this way from extractors which extract a constant fraction of the randomness. In light of this, we focus our attention on extractors which extract a constant fraction. The best such explicit extractor is that in [17], which uses a seed of length $O(\log^2 n)$.

¹Raz, Reingold, and Vadhan [16] give a general explicit transformation that transforms an extractor with constant error into an extractor with arbitrary small error while harming the other parameters only slightly more than is necessary. The exact dependence of our results on ϵ is presented in section 7.

²More formally, a family $E = \{E_n\}$ of extractors is defined given polynomially computable integer functions $d(n), m(n), k(n), \epsilon(n)$ such that for every n , $E_n : \{0, 1\}^n \times \{0, 1\}^{d(n)} \rightarrow \{0, 1\}^{m(n)}$ is a $(k(n), \epsilon(n))$ -extractor. The family is explicit in the sense that E_n can be computed in time polynomial in $n + d(n)$.

TABLE 2

Optimizing the seed length: $d = O(\log n)$. All the results are stated for constant ϵ . α is an arbitrary small constant.

Reference	min-entropy k	Output length m
[32]	$k = \Omega(n)$	$(1 - \alpha)k$
[27]	$k = n^{\Omega(1)}$	$k^{1-\alpha}$
[7]	$k = 2^{O(\sqrt{\log n})}$	$\Omega(\frac{k}{\log \log \log n})$
[7]	any k	$k^{1-\alpha}$
Thm. 1.6	any k	$\Omega(\frac{k}{\log n})$
optimal	any k	k

Concluding this presentation, we stress that while there are explicit constructions which are optimal in any one of the above two parameters (for arbitrary k), the cost is a polynomial loss in the other.

1.3. New results. We give two constructions, each optimal in one of the parameters and losing only a *polylogarithmic* factor in the other. Thus, both come closer to simultaneously optimizing both parameters. (We remark that a subsequent work [9] gives constructions that lose only a *constant* factor in the other parameter.) The results are stated for constant ϵ (see section 7 for the exact dependence on ϵ). In the first construction we extract any constant fraction of the initial randomness using a seed of length $O(\log n \cdot (\log \log n)^2)$. This improves the best previous such result, found in [17], which uses a seed of length $O(\log^2 n)$.

THEOREM 1.4. *For every n, k , and constant ϵ , there are explicit (k, ϵ) -extractors $\text{Ext} : \{0, 1\}^n \times \{0, 1\}^{O(\log n \cdot (\log \log n)^2)} \rightarrow \{0, 1\}^{(1-\alpha)k}$, where $\alpha > 0$ is an arbitrary constant.³*

Using [29], we get the following corollary,⁴ which also improves the previous best construction which extract all the randomness by [17].

COROLLARY 1.5. *For every n, k , and constant ϵ , there are explicit (k, ϵ) -extractors $\text{Ext} : \{0, 1\}^n \times \{0, 1\}^{O(\log n \cdot (\log \log n)^2 \cdot \log k)} \rightarrow \{0, 1\}^k$.*

Our second construction uses the optimal seed length (that is, $O(\log n)$) to extract $m = \Omega(k/\log n)$ bits; this improves the best previous result of [7] which could only extract $m = k^{1-\alpha}$ bits.

THEOREM 1.6. *For every n, k , and constant ϵ , there are explicit (k, ϵ) -extractors $\text{Ext} : \{0, 1\}^n \times \{0, 1\}^{O(\log n)} \rightarrow \{0, 1\}^{\Omega(k/\log n)}$.*

Using [7], we get the following corollary (in which the “loss” depends only on k).⁵

COROLLARY 1.7. *For every n, k , and constant ϵ , there are explicit (k, ϵ) -extractors $\text{Ext} : \{0, 1\}^n \times \{0, 1\}^{O(\log n)} \rightarrow \{0, 1\}^{\Omega(k/(\log k \cdot \log \log k))}$.*

³We remark that our construction requires $k \geq 8 \log^5 n$. Nevertheless, theorem 1.4 follows as stated because the case of $k < 8 \log^5 n$ was already covered in [7]. This is also the case in the next theorems.

⁴In fact, the version of Theorem 1.4 stated above does not suffice to conclude the corollary. To use the method of [29], we need a version in which the error is $1/\text{polylog} n$, which follows from our analysis; see section 7.

⁵Impagliazzo, Shaltiel, and Wigderson [7] show that an extractor $\text{Ext} : \{0, 1\}^{k^{O(1)}} \times \{0, 1\}^d \rightarrow \{0, 1\}^m$ can be used to construct an extractor $\text{Ext} : \{0, 1\}^n \times \{0, 1\}^{d+O(\log n)} \rightarrow \{0, 1\}^{\Omega(m/\log \log k)}$ for any n .

1.4. Condensers. The main technical contribution of this paper consists of constructions of various “condensers.”⁶ A condenser is a generalization of an extractor.

DEFINITION 1.8. *A (k, k', ϵ) -condenser is a function $Con : \{0, 1\}^n \times \{0, 1\}^d \rightarrow \{0, 1\}^{n'}$ such that for every k -source X of length n , the distribution $Con(X, U_d)$ is ϵ -close to a k' -source.*

Note that an extractor is a special case of a condenser when $n' = k'$. Condensers are most interesting when $k'/n' > k/n$ (that is, when the entropy rate of the output distribution is larger than that of the initial distribution). Such condensers can be used to construct extractors via repeatedly condensing the source until an entropy rate 1 is achieved. In this paper we give a new construction of condensers. One possible setting of the parameters gives that, for constant $\epsilon > 0$ and every n and k with $k \geq 8 \log^5 n$, there is an explicit $(k, \Omega(k), \epsilon)$ -condenser $Con : \{0, 1\}^n \times \{0, 1\}^{O(\log n \cdot \log \log n)} \rightarrow \{0, 1\}^{k \log n}$. The exact parameters can be found in section 5.

Somewhat surprisingly, an important component in the construction of the condenser above is a condenser that does not condense at all! That is one in which the entropy rate of the output distribution is *smaller* than that of the initial distribution. The usefulness of such objects in constructing extractors is demonstrated in [13] (see also [23, 32]). We refer to such condensers as “block extraction schemes” and elaborate on their role in extractor (and condenser) constructions in section 1.6. In this paper we give a construction of a block extraction scheme that uses a seed of length $O(\log \log n)$. The exact parameters can be found in section 3.

1.5. Transforming general extractors into strong extractors. Speaking informally, a strong extractor is an extractor in which the output distribution is independent of the seed.⁷ In some applications of extractors, it is beneficial to have the strong version. Most extractor constructions naturally lead to strong extractors, yet some (with examples being the constructions of [24, 7] and this paper) are not strong or are difficult to analyze as strong. We solve this difficulty by giving a general explicit transformation which transforms any extractor into a strong extractor with essentially the same parameters. Exact details are given in section 8.

1.6. Technique. In this section we attempt to explain the main ideas in this paper without delving into technical details.

High level overview. In contrast to the latest papers on extractors [27, 17, 7] we do not use Trevisan’s paradigm. Instead, we revisit [13] and attempt to construct block-sources (a special kind of source which allow very efficient extraction). Following [23, 12], we observe that, when failing to produce a block-source, the method of [13] “condenses” the source. This enables us to use a “win-win” case analysis, as in [6, 7], which eventually results in a construction of a condenser. Our extractors are then constructed by repeated condensing. A critical component in obtaining improved parameters is derandomizing the construction of block-sources of [13].

Block-sources. We begin by describing a special kind of source called block-sources (a precise definition appears in Definition 2.3) that allow for the construction of extractors with very efficient parameters. Consider a source distribution X that

⁶The notion of condensers was also used in [15]. While similar in spirit, that paper deals with a completely different set of parameters and uses different techniques. The notion of condensers also comes up in subsequent work [25, 2, 9] as a tool for constructing extractors and expander graphs.

⁷In the original paper [13] strong versions of extractors were initially defined and constructed, and the notion of a “nonstrong” extractor was later given by Ta-Shma [24].

is composed of two independent concatenated distributions $X = (X_1, X_2)$, where X_1 is a k_1 -source and X_2 is a k_2 -source. Extractors for this special scenario (which are called block-source extractors) can be constructed by composing two extractors. An extractor with optimal seed length can be used to extract random bits from X_2 , and these bits (being independent of X_1), can be used as seed to extract all the randomness from X_1 using an extractor with large output. (Note that with today's best extractors this construction uses the optimal seed length to extract all the randomness from X_1 , as long as k_2 is at least $\text{polylog}(n)$.) The requirement that X_1 and X_2 be independent could be relaxed in the following way (which was suggested in [3]). Intuitively, it is sufficient that X_1 "contains" k_1 random bits, and X_2 "contains" k_2 random bits even conditioned on X_1 . Such sources are called block-sources.⁸ Thus, extracting randomness from general sources can be achieved by giving a construction which uses few random bits to transform a general source into a block-source and by using a block-source extractor.

This approach was suggested by Nisan and Zuckerman in [13]. They constructed a "block extraction scheme." That is a condenser that given an arbitrary source X , uses few random bits to produce a new source B (called a block) which is shorter than the initial source, and contains a large fraction of the initial randomness. This means that the distribution (B, X) meets the first requirement of block-sources, i.e., the first block contains randomness. Intuitively, to meet the second requirement one should give an upper bound on the amount of randomness contained in B , and conclude that there is some randomness in X which is not contained in B . However, in the construction of Nisan and Zuckerman such an upper bound can be achieved only "artificially" by setting the parameters so that the length of B is smaller than k . This indeed gives that B does not "steal all the entropy" in the source. However, this approach has a costly effect. In the analysis of Nisan and Zuckerman the amount of randomness that is guaranteed to be in B is proportional to its length. Thus, decreasing the length of B reduces the amount of entropy that can be guaranteed. In particular, when $k < \sqrt{n}$ and the length of B is chosen in the way explained above, it may be the case that B contains no randomness. As a result, the extractors of [13] do not work when $k < \sqrt{n}$.

Condensing by a "win-win" analysis. A way to get around this problem was suggested in [23, 12]. We now explain a variant of the argument in [12] that we use in our construction. Recall that we want to obtain a large block B that does not "steal all the entropy." An important observation is that the case in which the block extraction scheme fails to produce such a block is also good in the sense that it means that B "stole all the entropy from the source." As B is shorter than the initial source, it follows that it is more condensed. We now explain this approach in more detail.

Suppose we use the block extraction scheme to produce a large block B (say, of length $n/2$) and consider the distribution (B, X) . Recall that, for our purposes, to get a useful block-source it suffices that X contains very few random bits that are not contained in B . An important observation is that when the procedure above fails to construct a block-source, this happens because (almost) all the randomness "landed" in B . In this case, we obtained a block that is more condensed than the initial source X . (It has roughly the same amount of randomness and half the length.)

Using this idea repeatedly, at each step either we construct a block-source (from which we can extract randomness) or we condense the source. There is a limit to

⁸More precisely, a (k_1, k_2) -block-source is a distribution $X = (X_1, X_2)$ such that X_1 is a k_1 -source, and for every possible value x_1 of X_1 , the distribution of X_2 , conditioned on the event that $X_1 = x_1$, is a k_2 -source.

how much we can condense the source. Certainly, when the length reduces below the original entropy k , no further condensing is possible. This means that by running this procedure repeatedly enough times we eventually obtain a block-source.

The outcome of the procedure above consists of several “candidate” distributions, where one of them is a block-source. Not knowing which is the “right one,” we run block-source extractors on all of them (using the same seed). We know that one of the distributions we obtain is close to uniform. It turns out that the number of candidate distributions is relatively small (about $\log(n/k)$). Consider the distribution obtained by concatenating the output distributions of the block-source extractors. This distribution contains a portion which is (close to being) uniformly distributed on roughly k bits and thus has entropy close to that of the initial source. Moreover, the distribution is on strings of length not much larger than k (the length is roughly $k \log(n/k)$). We conclude that the aforementioned distribution is more condensed than the initial source and that the procedure described above is a condenser!

We can now obtain an extractor construction by repeatedly condensing the source (using fresh randomness in each iteration) until it becomes close to uniform. However, it turns out that the parameters of the constructed condenser are not good enough to yield a good extractor. Our actual construction of condensers is achieved by using the procedure above with an improved version of the block extraction scheme of Nisan and Zuckerman.

Improved block extraction. Let us delve into the parameters. The block extraction scheme of Nisan and Zuckerman spends $O(\log n)$ random bits when producing a block B of length $n/2$ and guarantees that B is an $\Omega(k/\log(n/k))$ -source. This turns out to be too costly and we cannot run our condenser construction.

One problem is that the number of random bits used by the block extraction scheme is too large for our purposes. Note that the block extraction scheme already spends $O(\log n)$ random bits. Using the strategy described above, we will need to run it roughly $\log(n/k)$ times, which results in a large seed length ($\log n \cdot \log(n/k)$). We overcome this problem by derandomizing the construction of Nisan and Zuckerman. We reduce the number of random bits used from $\log n$ to $\log \log n$, which allows us to run it a larger number of times while still obtaining short seed length.

A second problem is that we want the block B to contain a constant fraction of the initial randomness k . The analysis of Nisan and Zuckerman guarantees only that the block B contains $\Omega(k/\log(n/k))$ random bits. Note that while this quantity is smaller than we want, it does achieve the goal when k is a constant fraction of n . We introduce another modification to the construction of Nisan and Zuckerman in order to increase the randomness in B . The idea is to reduce the case of $k = o(n)$ to the case of $k = \Omega(n)$. This is done by error correcting the source prior to using the block extraction scheme. We give a nonconstructive argument to show that every error corrected random source has a “piece” of length k which is an $\Omega(k)$ -source. When analyzing the scheme, we use the analysis of Nisan and Zuckerman on this piece. Intuitively, this enables the analysis of the block extraction scheme to be carried out on the condensed piece, where it performs best.

1.7. Organization of the paper. Section 2 includes some preliminaries. In section 3 we construct a block extraction scheme. In section 4 we use the method of [12] to show that when using the block extraction scheme either we get a block-source or we condense the source. In section 5 we run the block extraction scheme recursively and obtain condensers. In section 6 we use the condensers to construct extractors. Section 7 gives the exact dependence of our extractors on the error parameter ϵ . In

section 8 we show how to transform arbitrary extractors into strong extractors.

2. Preliminaries.

2.1. Probability distributions. Given a probability distribution P , we use the notation $\Pr_P[\cdot]$ to denote the probability of the relevant event according to the distribution P . We sometimes fix some probability space (namely, a set Ω and a distribution over Ω) and then use the notation $\Pr[\cdot]$ to denote the probability of events in this probability space. We use $\Pr[E_1, E_2]$ to denote $\Pr[E_1 \cap E_2]$.

We need the following notion of “collision probability.”

DEFINITION 2.1. *For a distribution P over Ω , define $C(P) = \sum_{\omega \in \Omega} P(\omega)^2$. In other words, $C(P)$ is the probability that two independent samples from P give the same outcome. We refer to $C(P)$ as the collision probability of P .*

We need the following useful lemma.

LEMMA 2.2. *Let V be ρ -close to a k -source. Define $L = \{v \mid \Pr[V = v] > 2^{-(k-1)}\}$. It follows that $\Pr(V \in L) < 2\rho$.*

Proof. Let V' be a k -source such that V and V' are ρ -close. We have that $|\Pr_V(L) - \Pr_{V'}(L)| < \rho$. However, V' assigns small probability to all elements in L , whereas V assigns large probability to these elements. This gives that $\Pr_V(L) - \Pr_{V'}(L) > \Pr_{V'}(L)$, which means that $\Pr_{V'}(L) < \rho$. Using the inequality above, we get that $\Pr_V(L) < 2\rho$. \square

2.2. Block-sources. Block-sources are random sources which have a special structure. The notion of block-sources was defined in [3].

DEFINITION 2.3 (see [3]). *Two random variables (X_1, X_2) form a (k_1, k_2) -block-source if X_1 is a k_1 -source, and for every possible value x_1 of X_1 , the distribution of X_2 , given that $X_1 = x_1$, is a k_2 -source.*

Block-source extractors are extractors which work on block-sources.

DEFINITION 2.4. *A (k, t, ϵ) -block-source extractor is a function $\text{Ext} : \{0, 1\}^{n_1} \times \{0, 1\}^{n_2} \times \{0, 1\}^d \rightarrow \{0, 1\}^m$ such that for every (k, t) -block-source (X_1, X_2) (where X_1, X_2 are of length n_1, n_2 , respectively), the distribution $\text{Ext}(X_1, X_2, U_d)$ is ϵ -close to U_m .*

Block-sources allow the following composition of extractors.

LEMMA 2.5 (implicit in [13]; also appears in [23]). *If there exist an explicit (k, ϵ_1) -extractor $\text{Ext}_1 : \{0, 1\}^{n_1} \times \{0, 1\}^{d_1} \rightarrow \{0, 1\}^m$, and an explicit (t, ϵ_2) -extractor $\text{Ext}_2 : \{0, 1\}^{n_2} \times \{0, 1\}^{d_2} \rightarrow \{0, 1\}^{d_1}$, then there exists an explicit $(k, t, \epsilon_1 + \epsilon_2)$ -block-source extractor $\text{Ext} : \{0, 1\}^{n_1} \times \{0, 1\}^{n_2} \times \{0, 1\}^{d_2} \rightarrow \{0, 1\}^m$, where $\text{Ext}(x_1, x_2; y) = \text{Ext}_1(x_1, \text{Ext}_2(x_2, y))$.*

Following [13, 23, 32], we can use the above theorem to compose two extractors: one which optimizes the seed length and another which optimizes the output length. The resulting block-source extractor will “inherit” the nice properties of both its component extractors. Particularly, taking Ext_1 to be the extractor of [17] and Ext_2 to be the extractor of [7], we get the following block-source extractor.

COROLLARY 2.6. *For all integers $n_1 \leq n_2$, k , and $t \geq \log^4 n_1$, there is an explicit $(k, t, \frac{1}{n_1} + \frac{1}{n_2})$ -block-source extractor $BE : \{0, 1\}^{n_1} \times \{0, 1\}^{n_2} \times \{0, 1\}^{O(\log n_2)} \rightarrow \{0, 1\}^k$.*

Thus, to construct extractors which achieve short seed length and large output simultaneously, it suffices to use few random bits and convert any k -source into a $(k', \log^4 n)$ -block-source such that k' is not much smaller than k .

This turns out to be a tricky problem. No such scheme (efficient in terms of random bits spent) is known when $k < \sqrt{n}$.

2.3. Error correcting codes. Our construction uses error correcting codes.

DEFINITION 2.7. *An error correcting code with distance d is a function $EC : \{0, 1\}^n \rightarrow \{0, 1\}^{\bar{n}}$ such that for every $x_1, x_2 \in \{0, 1\}^n$ such that $x_1 \neq x_2$, $d_{\text{Hamming}}(EC(x_1), EC(x_2)) \geq d$. (Here $d_{\text{Hamming}}(z_1, z_2)$ denotes the Hamming distance between z_1, z_2). An error correcting code is explicit if EC can be computed in polynomial time.*

We use the following construction of error correcting codes.

THEOREM 2.8 (see [8]). *There exist constants $0 < b < a$ and an explicit error correcting code $EC : \{0, 1\}^n \rightarrow \{0, 1\}^{an}$ with distance bn .*

2.4. Almost l -wise independent distributions. We use the following notion of efficiently constructible distributions.

DEFINITION 2.9. *Call a distribution P on n bits, polynomially constructible using $u(n)$ bits,⁹ if there exists an algorithm $A : \{0, 1\}^{u(n)} \rightarrow \{0, 1\}^n$ which runs in time polynomial in n such that the distribution $A(Y)$, where Y is chosen uniformly from $\{0, 1\}^{u(n)}$, is identical to P .*

Naor and Naor [10] defined “almost l -wise independent distribution.”

DEFINITION 2.10 (see [10]). *A distribution (P_1, \dots, P_n) over $\{0, 1\}^n$ is said to be (ϵ, l) -wise dependent with mean p if for every subset $\{i_1, \dots, i_l\}$ of $[n]$, the distribution $(P_{i_1}, \dots, P_{i_l})$ is ϵ -close to the distribution over l bit strings, where all bits are independent and each of them takes the value 1 with probability p .*

Naor and Naor showed that almost l -wise independent distributions can be constructed using very few random bits.

THEOREM 2.11 (see [10]). *For every n, l , and ϵ , an (ϵ, l) -wise dependent distribution with mean $1/2$ is polynomially constructible using $O(\log \log n + l + \log(1/\epsilon))$ bits.*

We require distributions that are almost l -wise independent with mean different than $1/2$. Nevertheless, it is very easy to construct such distributions from Theorem 2.11.

COROLLARY 2.12. *For every n, ϵ , and q , an $(\epsilon, 2)$ -wise dependent distribution with mean 2^{-q} is polynomially constructible using $O(\log \log n + q + \log(1/\epsilon))$ bits.*

Proof. We use Theorem 2.11 to construct a distribution that is $(\epsilon, 2q)$ -wise dependent with mean $1/2$ over qn bits. Note that this costs $O(\log \log(qn) + q + \log(1/\epsilon)) = O(\log \log n + q + \log(1/\epsilon))$ bits. We denote these bits by P_1, \dots, P_{qn} . We divide them into n blocks of length q and define n bits P'_1, \dots, P'_n as follows: P'_i is set to “one” if and only if all the bits in the i th block are “one.” In particular, each P'_i is a function of the bits in the i th block. It follows that the distribution P'_1, \dots, P'_n is $(\epsilon, 2)$ -wise dependent. \square

Given (X_1, \dots, X_n) that form a $(0, 2)$ -wise dependent distribution, Chebyshev’s inequality gives that for every $0 < \lambda < 1$,

$$\Pr \left(\left| \sum_{1 \leq i \leq n} X_i - pn \right| > \lambda pn \right) < \frac{1}{\lambda^2 pn}.$$

The same argument can be applied to $(\epsilon, 2)$ -wise dependent distributions and gives the following.

⁹Naturally, one should speak about a sequence $P = \{P_n\}$ of distributions for this to make sense.

LEMMA 2.13 (see [10]). *If (X_1, \dots, X_n) is an $(\epsilon, 2)$ -wise dependent distribution with mean p , then for every $0 < \lambda < 1$,*

$$\Pr \left(\left| \sum_{1 \leq i \leq n} X_i - pn \right| > \lambda pn \right) < O \left(\frac{1}{\lambda^2 pn} + \sqrt{\epsilon} \right)$$

as long as $\epsilon < \frac{\lambda^4 p^4}{25}$.

3. Improved block extraction. Some constructions of block-sources from general sources [13, 23, 32] rely on a building block called a “block extraction scheme.” In our terminology a block extraction scheme is a condenser. Nevertheless, we choose to redefine this notion, as it is more convenient to present the parameters in a different way.

DEFINITION 3.1. *Let n, k , and r be integers and ρ, γ be numbers. A function $B : \{0, 1\}^n \times \{0, 1\}^d \rightarrow \{0, 1\}^{n/r}$ is a (k, ρ, γ) -block extraction scheme if it is a $(k, \gamma \cdot (\frac{k}{r}), \rho)$ -condenser.*

In other words, a block extraction scheme takes a k -source of length n and uses a seed to produce a distribution (which we call a block) of length n/r . The parameter γ measures the entropy rate of the new distribution in terms of the entropy rate of the initial distribution. For example, when $\gamma = 1$ this means that the two distributions have the same rate and the block extraction scheme “preserves” the entropy rate in the source. In this section, we discuss constructions in which $\gamma < 1$, meaning that the entropy rate in the output distribution is *smaller* than that of the initial distribution.

Using this notation, Nisan and Zuckerman proved the following theorem.

LEMMA 3.2 (see [13]). *There exists a constant $c > 0$ such that for every n, k, r , and $\rho \geq 2^{-ck/r}$ there is an explicit $(k, \rho, \Omega(\frac{1}{\log n/k}))$ -block extraction scheme $B : \{0, 1\}^n \times \{0, 1\}^{O(\log n \log 1/\rho)} \rightarrow \{0, 1\}^{n/r}$.*

We prove the following lemma, which improves Lemma 3.2 for some choice of parameters, namely, for $1 < r \leq \log n$ and $\rho = 1/\log^{O(1)} n$.

LEMMA 3.3. *There exists a constant $c > 0$ such that for every n, k, r , and $\rho \geq c\sqrt{\frac{\rho}{k}}$ there is an explicit $(k, \rho, \Omega(1))$ -block extraction scheme $B : \{0, 1\}^n \times \{0, 1\}^{O(\log \log n + \log(1/\rho) + \log r)} \rightarrow \{0, 1\}^{n/r}$.*

Lemma 3.3 improves Lemma 3.2 in the following two respects (as long as one settles for small $r \leq \log n$ and large $\rho > 1/\log^{O(1)} n$):

1. We reduce the number of random bits spent by the block extraction scheme.

In [13] the number of random bits is logarithmic in n , whereas in Lemma 3.3 the number of random bits is double logarithmic in n .

This is achieved by derandomizing the proof of Nisan and Zuckerman using almost l -wise independent distributions. In section 3.1 we describe the property that a distribution needs to have to allow for the Nisan–Zuckerman analysis, and we construct a small sample space with this property.

2. We increase the amount of randomness guaranteed in the output block. In [13] the amount of randomness guaranteed in the output block B is $\Omega(\frac{k}{r \log(n/k)})$.

Lemma 3.3 guarantees that B contains $\Omega(\frac{k}{r})$ random bits.

Note that the two quantities are the same when $k = \Omega(n)$. Indeed, our improvement is achieved by reducing the case of $k = o(n)$ to that of $k = \Omega(n)$.

We start from a k -source with $k = o(n)$. In section 3.3 we show that once a random source is error corrected, there are some k indices (to the error corrected source) which induce an $\Omega(k)$ -source. This induced source has a

constant entropy rate. When applying the argument of Nisan and Zuckerman, our analysis concentrates on this source which allows us to guarantee that the block contains more randomness. The exact analysis is given in section 3.4. The remainder of this section is devoted to proving Lemma 3.3.

3.1. The analysis of Nisan and Zuckerman. The block extraction scheme of Nisan and Zuckerman is obtained by restricting the source to some subset of the indices which is selected using few random bits. More precisely, they construct a small sample space of subsets of $[n]$ (having a property that we immediately describe) and prove that the distribution obtained by sampling an element from a k -source and restricting it to the indices in a random set from the sample space contains a large fraction of the initial randomness. In this section we construct a smaller such sample space which enables us to spend less random bits to construct a block extraction scheme. We now describe the property used by Nisan and Zuckerman. Intuitively, a k -source has k random bits “hidden” somewhere.

DEFINITION 3.4. *Let n, k, r , and δ be parameters. A distribution S over subsets of $[n]$ is called r -intersecting for sets of size k with error probability δ if for every $G \subseteq [n]$ with $|G| \geq k$, $\Pr_S(|S \cap G| < \frac{k}{8r}) < \delta$.*

The following is implicit in [13].

LEMMA 3.5 (see [13]). *There exists some constant $c > 0$ such that if X is a k -source on $\{0, 1\}^n$ and S is a distribution over subsets of $[n]$ which is r -intersecting for sets of size $\frac{ck}{\log(n/k)}$ with error probability δ , then the distribution $X|_S$ (obtained by sampling x from X and s from S and computing $x|_s$) is $(4\sqrt{\delta} + 2^{-\Omega(k)})$ -close to an $\Omega(\frac{k}{r \log(n/k)})$ -source.*

Nisan and Zuckerman use a construction based on $O(\log(1/\delta))$ -wise independence to prove the following lemma.

LEMMA 3.6 (see [13]). *For every n, k, r , and $\delta > 2^{-O(k/r)}$ there is a distribution over subsets of $[n]$ that are of size n/r and this distribution is r -intersecting for sets of size k with error probability δ . Furthermore, the distribution is polynomially constructible using $O(\log n \cdot \log(1/\delta))$ bits.*

Using Lemma 3.5, this immediately implies the block extraction scheme of Lemma 3.2. We will be mostly interested in the case when r is small (say $r \leq \log n$) and δ is large, (say $\delta \geq \log^{-O(1)} n$). For this setup we can save random bits and make the dependence on n double logarithmic.

LEMMA 3.7. *There exists a constant $c > 0$ such that for every n, k, r , and $\delta > cr/k$, there is a distribution over subsets of $[n]$ that are of size n/r and this distribution is r -intersecting for sets of size k with error probability $\delta > 0$. Furthermore, the distribution is polynomially constructible using $O(\log \log n + \log r + \log(1/\delta))$ bits.*

We prove this lemma in the following subsection.

3.2. A small sample space for intersecting large sets. We now prove Lemma 3.7. We view distributions over n bit strings as distributions over subsets of $[n]$. More specifically, we identify the n bit values (W_1, \dots, W_n) with the set $\{i | W_i = 1\}$. We construct random variables (W_1, \dots, W_n) with the following properties:

- For every $1 \leq i \leq n$, $\Pr(W_i = 1) \approx 1/2r$.
- For every set $G \subseteq [n]$ with $|G| \geq k$, the probability that the sum of the W_i 's for $i \in G$ is far from the expected $|G|/2r$ is small. (It is important to note that we allow the “small probability” to be quite large, since we are shooting for large δ .)

Note that the second condition gives both the intersecting property and the fact that the selected sets are rarely of size larger than n/r (by considering $G = [n]$). We are interested in constructing such a distribution using as few random bits as possible. A pairwise independent distribution has these properties but takes $\log n$ random bits to construct. We can do better by using the “almost l -wise independent” distributions of [10].

CONSTRUCTION 3.8. *Let q be an integer such that $1/4r < 2^{-q} \leq 1/2r$ and $\epsilon = \min(c\delta^2, c/r^4)$, where $c > 0$ is a constant that will be determined later. Let $W = (W_1, \dots, W_n)$ be the $(\epsilon, 2)$ -wise dependent distribution with mean 2^{-q} guaranteed in Corollary 2.12. Note that this requires $O(\log \log n + \log r + \log(1/\epsilon))$ random bits.*

The next lemma follows.

LEMMA 3.9. *There exist constants $c, c' > 0$ such that when using construction 3.8 with the constant c , for every $\delta > c' \cdot r/k$ the distribution W has the following properties:*

1. *For every set $G \subseteq [n]$ such that $|G| \geq k$, $\Pr(\sum_{i \in G} W_i < \frac{k}{8r}) < \frac{\delta}{3}$.*
2. $\Pr(\sum_{1 \leq i \leq n} W_i > \frac{n}{r}) < \frac{\delta}{3}$.

Proof. We use Lemma 2.13 to deduce both parts of the lemma and we use the fact that the W_i 's are $(\epsilon, 2)$ dependent with mean $p = 1/2^q$, where $1/4r < p \leq 1/2r$. We start by proving the first part. For this part, we set $\lambda = 1/2$ and assume without loss of generality that $|G| = k$. Note that

$$\left\{ \sum_{i \in G} W_i < \frac{k}{8r} \right\} \subseteq \left\{ \sum_{i \in G} W_i < \frac{k}{2 \cdot 2^q} \right\} \subseteq \left\{ \left| \sum_{i \in G} W_i - \frac{k}{2^q} \right| > \frac{\lambda k}{2^q} \right\}.$$

Thus, it suffices to bound the probability of the latter event. To meet the condition in Lemma 2.13 we need to make sure that $\epsilon < \frac{\lambda^4 p^4}{25} = \Theta(\frac{1}{r^4})$. The requirement that $\epsilon < c/r^4$ takes care of this condition by choosing a sufficiently small constant $c > 0$. Applying Lemma 2.13, we get that the probability of deviation from the mean is bounded by $O(r/k + \delta\sqrt{c})$. We have that $r/k \leq \delta/c'$. We can set c' to be large enough so that $O(r/k + \delta\sqrt{c}) \leq \delta/6 + O(\delta\sqrt{c})$. This is bounded from above by $\delta/3$ for small enough $c > 0$.

The proof of the second item is similar. We choose $\lambda = 1$ and note that

$$\left\{ \sum_{1 \leq i \leq n} W_i > \frac{n}{r} \right\} \subseteq \left\{ \sum_{1 \leq i \leq n} W_i > \frac{2n}{2^q} \right\} \subseteq \left\{ \left| \sum_{1 \leq i \leq n} W_i - \frac{n}{2^q} \right| > \frac{\lambda n}{2^q} \right\}.$$

Using the fact that $n \geq k$, we can repeat the computations above and conclude that the probability of this event is also bounded by $O(\delta\sqrt{c})$. The lemma follows by choosing a sufficiently small $c > 0$. \square

We are ready to prove Lemma 3.7.

Proof of Lemma 3.7. The first item of Lemma 3.9 shows that W is a distribution over subsets of $[n]$ that is r -intersecting for sets of size k with error probability $\delta/3$. The second item shows that W could be transformed into a distribution over subsets of size exactly n/r without changing it by much. This change is done by adding arbitrary indices to the set if its size is smaller than n/r and deleting arbitrary indices if its size is larger than n/r . Adding indices will not spoil the intersecting property, and the probability that we need to delete indices is bounded by $\delta/3$. The lemma follows. \square

3.3. Error corrected random sources. In this subsection we develop another tool required for the proof of Lemma 3.3 and show that if we apply an error correcting

code to an arbitrary k -source, we obtain a k -source which has k indices which induce an $\Omega(k)$ -source.

In the remainder of this section we fix a, b , and EC to be as in Theorem 2.8. For a vector $x \in \{0, 1\}^n$ and a set $T \subseteq [n]$, we use $x|_T$ to denote the restriction of x to T .

LEMMA 3.10. *Let X be a k -source on $\{0, 1\}^n$. There exists a set $T \subseteq [an]$ of size k such that $EC(X)|_T$ is an $\Omega(k)$ -source.*

Lemma 3.10 is an immediate corollary of Lemma 3.11 which was mentioned to us by Russell Impagliazzo. A very similar argument also appears in [23].

LEMMA 3.11 (see [5]). *Let X be a k -source on $\{0, 1\}^n$. For every v , there exists a set $T \subseteq [an]$ of size v such that $EC(X)|_T$ is a $\frac{1}{2} \cdot \log 1/(2^{-k} + (1 - \frac{b}{a})^v)$ -source.*

The following fact states that if a distribution has low collision probability, then it has high min-entropy. This follows because a distribution with low min-entropy has an element which gets large probability. This element has a large chance of appearing in two consecutive independent samples.

FACT 3.12. *If $C(P) \leq 2^{-k}$, then P is a $(k/2)$ -source.*

Our goal is to show that there exists a subset of $[an]$ on which the error corrected source has low collision probability. We will show that a random (multi-) set has this property.

Proof of Lemma 3.11. Consider the following probability space: X_1, X_2 are independently chosen from the distribution X , and $T = (I_1, \dots, I_v)$ is chosen independently, where each I_j is uniformly distributed in $[an]$. Consider the following event: $B = \{EC(X_1)|_T = EC(X_2)|_T\}$. We first show that the probability of B is small:

$$(3.1) \quad \Pr(B) = \Pr(B|X_1 = X_2) \Pr(X_1 = X_2) + \sum_{a_1 \neq a_2} \Pr(B|X_1 = a_1, X_2 = a_2) \Pr(X_1 = a_1, X_2 = a_2).$$

X is a k -source, and therefore $\Pr(X_1 = X_2) \leq 2^{-k}$. For given $a_1 \neq a_2$, we know that the distance between $EC(a_1)$ and $EC(a_2)$ is at least bn . Thus, any of the I_j 's has a chance of $\frac{b}{a}$ to "hit" a coordinate where $EC(a_1)$ and $EC(a_2)$ disagree. Having chosen v such coordinates, the probability that none of them differentiate between $EC(a_1)$ and $EC(a_2)$ is bounded by $(1 - \frac{b}{a})^v$. Plugging this into (3.1) we get that

$$\Pr(B) \leq 2^{-k} + \left(1 - \frac{b}{a}\right)^v.$$

In the sample space we considered, T was chosen at random. Still, by averaging there exists a fixed value T' of the random variable T for which the above inequality holds. For this T' we have that the probability that independently chosen X_1 and X_2 have $EC(X_1)|_{T'} = EC(X_2)|_{T'}$ is small. In other words, we have that

$$C(EC(X)|_{T'}) \leq 2^{-k} + \left(1 - \frac{b}{a}\right)^v.$$

The lemma immediately follows from Fact 3.12. \square

3.4. Construction of the block extraction scheme. In this subsection we put everything together and prove Lemma 3.3. We are ready to define our block extraction scheme. Recall that EC is the error correcting code from Theorem 2.8 and a and b are the constants whose existence is guaranteed by that theorem.

CONSTRUCTION 3.13 (block extraction scheme). *Given n, k, r, ρ , and a constant e (which will be fixed later), let $d = O(\log \log n + \log r + \log(1/\rho))$ be the number of bits used by Lemma 3.7 to construct a distribution over subsets of $[an]$ that is ar -intersecting for sets of size ek with error probability $(\frac{\rho}{4})^2$. For $y \in \{0, 1\}^u$, let S_y be the set defined by y in the intersecting distribution. We now define*

$$B(x, y) = EC(x)|_{S_y}.$$

We are finally ready to prove Lemma 3.3.

Proof of Lemma 3.3. Let V denote the distribution $EC(X)$. Lemma 3.10 implies that there exists a set $T \subseteq [an]$ of size k such that $V|_T$ is an ηk -source (for some constant $\eta > 0$). Consider the distribution $S \cap T$ (the restriction of the intersecting distribution to the coordinates of T). Note that the distribution $S \cap T$ is a distribution over subsets of T . We claim that it has the same intersecting properties of S , namely, that $S \cap T$ is ar -intersecting for sets of size ek with error probability $(\frac{\rho}{4})^2$. (This follows from the definition, as every subset $G \subseteq T$ is in particular a subset of $[n]$.) We now use Lemma 3.5 on the source $V|_T$ using the intersecting distribution $S \cap T$. Let us first check that the conditions of Lemma 3.5 are met. We fix the constant e of Construction 3.13, setting $e = (c\eta)/(-\log \eta)$, where c is the constant from Lemma 3.5. The conditions of Lemma 3.5 are met since $V|_T$ is an ηk -source of length k and we have a distribution consisting of intersecting sets of size $ek = \frac{c(\eta k)}{\log(k/(\eta k))}$. We conclude from the lemma that $V|_{S \cap T}$ is ρ -close to an $\Omega(k/r)$ -source. We now claim that $V|_S$ is ρ -close to an $\Omega(k/r)$ -source. This is because adding more coordinates cannot reduce the entropy. The lemma follows, as we have shown that $B(X, U_d) = V|_S$ is ρ -close to an $\Omega(k/r)$ -source. \square

4. Partitioning to two “good” cases. Let B be the block extraction scheme constructed in the previous section and let X be a k -source. We consider the distribution $(B(X, Y), X)$ (where Y is a random seed that is independent of X). Following the intuition explained in section 1.6, we would like to argue that for every k -source X , at least one of the following good cases occurs:

- $(B(X, Y), X)$ is (close to) a block-source.
- $B(X, Y)$ is (close to) having a higher entropy rate than X ; that is, $B(X, Y)$ is more condensed than X .

In this section we show that, although the statement above does not hold as stated, we can prove a more technical result with the same flavor.

Remark 4.1. Here is a counterexample for the statement above assuming $k \leq n/2$. To make the example more simple, we assume that the block extraction scheme does not error correct the source prior to choosing a subset. Consider a source X which tosses a random bit b and, depending on the outcome, decides whether to sample from a distribution X_1 , in which the first k bits are random and the remaining $n - k$ bits are fixed, or from a distribution X_2 that is k -wise independent. X_1 corresponds to the first good case (and yields a block-source), as B is expected to hit about $k/2$ random bits. X_2 corresponds to the second (and yields a condensed block), as the block that B outputs contains $n/2$ bits and thus “steals all the randomness.” However, the “combination distribution” X does not satisfy any of the two good cases.

A way of getting around this problem was devised in [12]. The idea is to argue that the example in the remark above is the “worst possible” and show that any source X can be partitioned into two sources, where for each of them one of the good cases holds. To make this formal, we introduce the following notation.

DEFINITION 4.2 (conditioning random variables). *Fix some probability space. Let X be a random variable and E be an event with positive probability. We define the probability distribution of X conditioned on E , which we denote by $P_{(X|E)}$ as follows: For every possible value x in the range of X ,*

$$P_{(X|E)}(x) = \begin{cases} \Pr(X = x|E) & \Pr(X = x, E) > 0 \\ 0 & \Pr(X = x, E) = 0 \end{cases}$$

- For a random variable X and an event E we say that X is a k -source in E if $P_{(X|E)}$ is a k -source.
- For two random variables A, B and an event E , we say that the pair (A, B) is a (k_1, k_2) -block-source in E if $P_{((A,B)|E)}$ is a (k_1, k_2) -source.

We use the convention that if E has zero probability, then any random variable is a k -source (or (k_1, k_2) -block-source) in E .

With this notation, the precise statement is that, given some k -source X and uniformly distributed seed Y for the block extraction scheme, we can partition this probability space into three sets: The first has negligible weight and can be ignored. In the second, the block extraction scheme produces a block-source, and in the third, the block extraction scheme condenses the source. We now state this precisely.

For the remainder of this section we fix the following parameters:

- a k -source X over n bit strings,
- an (k, ρ, γ) block extraction scheme $B : \{0, 1\}^n \times \{0, 1\}^u \rightarrow \{0, 1\}^{n/r}$ for $r \geq 2$,
- a parameter t . (Intuitively, t measures how much randomness we want the second block of a block-source to contain.)

We now fix the following probability space that will be used in the remainder of this section. The probability space is over the set $\Omega = \{0, 1\}^n \times \{0, 1\}^u$ and consists of two independent random variables: X (the given k -source) and Y (uniformly distributed over $\{0, 1\}^u$).

LEMMA 4.3. *There exists a partition of $\{0, 1\}^n \times \{0, 1\}^u$ into three sets, BAD , BLK , CON , with the following properties:*

1. $\Pr(BAD) \leq 2(\rho + 2^{-t})$.
2. $(B(X, Y), X)$ is a $(\frac{\gamma k}{r} - t, t)$ -block-source in BLK .
3. $B(X, Y)$ is a $(k - 2t)$ -source in CON .

In the remainder of this section we use an idea similar to that in [12] to prove Lemma 4.3. This idea is to partition the elements in the probability space into three sets according to their “weight”: The “small weight” elements form the set CON . Intuitively the small weight elements induce a source of high min-entropy. The “medium-weight” elements form the set BLK . Intuitively the medium weight elements induce a source of medium min-entropy. Thus, they contain some (but not all!) of the min-entropy of the initial source. The fraction of “large weight” elements is bounded by ρ (the error parameter of the block extraction scheme). These elements form the set BAD and can be ignored because of their small fraction.

The following definition is motivated by the intuition above. (The partition of Lemma 4.3 will be based on the following partition.)

DEFINITION 4.4. *We partition $\Omega = \{0, 1\}^n \times \{0, 1\}^u$ according to the “weight” of the elements:*

$$\begin{aligned} LRG &= \{(x, y) \in \Omega \mid 2^{-(\frac{\gamma k}{r}-1)} < \Pr(B(X, Y) = B(x, y))\} \\ MED &= \{(x, y) \in \Omega \mid 2^{-(k-t)} < \Pr(B(X, Y) = B(x, y)) \leq 2^{-(\frac{\gamma k}{r}-1)}\} \\ SML &= \{(x, y) \in \Omega \mid \Pr(B(X, Y) = B(x, y)) \leq 2^{-(k-t)}\}. \end{aligned}$$

We prove the following lemma.

LEMMA 4.5. *The sets LRG, MED, and SML have the following properties:*

1. $\Pr(LRG) \leq 2\rho$.
2. $(B(X, Y), X)$ is a $(\frac{\gamma k}{r} - \log \frac{1}{\Pr(MED)} - 1, t)$ -block-source in MED.
3. $B(X, Y)$ is a $(k - (t + \log \frac{1}{\Pr(SML)}))$ -source in SML.

Proof of Lemma 4.5.

Proof of first item. We apply Lemma 2.2, choosing $V = B(X, Y)$. Note that V is guaranteed to be ρ -close to a $(\gamma k/r)$ -source and therefore, by the lemma, $\Pr(LRG) \leq 2\rho$.

Proof of second item. Note that we need to prove the following:

- For every $(x, y) \in MED$, $\Pr(B(X, Y) = B(x, y) | MED) \leq 2^{-(\frac{\gamma k}{r} - \log \frac{1}{\Pr(MED)} - 1)}$.
- For every $(x, y) \in MED$, $\Pr(X = x | B(X, Y) = B(x, y), MED) \leq 2^{-t}$.

For the first statement, we use the following inequality: For every two events E_1, E_2 ,

$$(4.1) \quad \Pr(E_1 | E_2) = \frac{\Pr(E_1 \cap E_2)}{\Pr(E_2)} \leq \frac{\Pr(E_1)}{\Pr(E_2)}.$$

Applying this rule on the first statement gives

$$\begin{aligned} \Pr(B(X, Y) = B(x, y) | MED) &\leq \frac{\Pr(B(X, Y) = B(x, y))}{\Pr(MED)} \\ &\leq \frac{2^{-(\frac{\gamma k}{r} - 1)}}{\Pr(MED)} \leq 2^{-(\frac{\gamma k}{r} - \log \frac{1}{\Pr(MED)} - 1)}. \end{aligned}$$

Here we used the definition of MED.

We now prove the second statement:

$$\Pr(X = x | B(X, Y) = B(x, y), MED) = \frac{\Pr(X = x, B(X, Y) = B(x, y), MED)}{\Pr(B(X, Y) = B(x, y), MED)}.$$

Note that whether a given pair (x, y) is in the set MED depends only on the value $B(x, y)$. Thus, $\{B(X, Y) = B(x, y), MED\} = \{B(X, Y) = B(x, y)\}$ because when $B(X, Y) = B(x, y)$ for $(x, y) \in MED$, we already know that $(X, Y) \in MED$. Thus,

$$= \frac{\Pr(X = x, B(X, Y) = B(x, y))}{\Pr(B(X, Y) = B(x, y))} \leq \frac{\Pr(X = x)}{\Pr(B(X, Y) = B(x, y))} \leq \frac{2^{-k}}{2^{-(k-t)}} = 2^{-t},$$

where the last inequality follows from the fact that X is a k -source and from the definition of MED.

Proof of the third item. We need to prove that for $(x, y) \in SML$,

$$\Pr(B(X, Y) = B(x, y) | SML) \leq 2^{-(k - (t + \log \frac{1}{\Pr(SML)}))}.$$

The proof is similar to the proof of the first part in the second item. More precisely, we use the rule (4.1) above. We argue that

$$\begin{aligned} \Pr(B(X, Y) = B(x, y) | SML) &\leq \frac{\Pr(B(X, Y) = B(x, y))}{\Pr(SML)} \\ &\leq \frac{2^{-(k-t)}}{\Pr(SML)} \leq 2^{-(k - (t + \log \frac{1}{\Pr(SML)}))} \quad \square \end{aligned}$$

We are now ready to prove Lemma 4.3.

Proof of Lemma 4.3. We need to slightly change the partition above. The sets LRG, MED , and SML are almost the partition we want. We need only avoid a setup in which the sets MED or SML are too small, since in this case the effect of conditioning is too costly. Still, if one of the sets is very small, we can safely add it to the “bad” elements and ignore it. This is the intuition behind the following partition, which partitions $\{0, 1\}^n \times \{0, 1\}^u$ into three sets:

1. The set BAD will contain all $(x, y) \in LRG$. It will also contain all $(x, y) \in SML$ if $\Pr(SML) < 2^{-t}$, and all $(x, y) \in MED$ if $\Pr(MED) < 2^{-t}$.
2. The set BLK (which corresponds to the set MED) contains all $(x, y) \in MED$ if $\Pr(MED) \geq 2^{-t}$. (Thus, BLK is empty if $\Pr(MED) < 2^{-t}$.)
3. The set CON (which corresponds to the set SML) contains all $(x, y) \in SML$ if $\Pr(SML) \geq 2^{-t}$. (Thus, BLK is empty if $\Pr(SML) < 2^{-t}$.)

Lemma 4.3 follows from Lemma 4.5. \square

5. Constructing condensers. In this section we use a win-win analysis as outlined in section 1.6 to construct a condenser. The main theorem of this section is the following.

THEOREM 5.1. *For every n and k such that $k \geq 8 \log^5 n$ and $2 \leq r \leq \log^2 n$, there is an explicit $(k, \Omega(k/r), 1/\log^2 n)$ -condenser $Con : \{0, 1\}^n \times \{0, 1\}^{O(\frac{\log(n/k) \cdot \log \log n}{\log r} + \log n)} \rightarrow \{0, 1\}^{\frac{k \log(n/k)}{r \log r}}$.*

It is helpful to consider two particular corollaries. For the first one, we choose $r = 2$. This gives that the condenser maintains a constant fraction of the initial randomness.

COROLLARY 5.2. *For every n and k such that $k \geq 8 \log^5 n$, there is an explicit $(k, \Omega(k), 1/\log^2 n)$ -condenser $Con : \{0, 1\}^n \times \{0, 1\}^{O(\log(n/k) \log \log n + \log n)} \rightarrow \{0, 1\}^{O(k \log(n/k))}$.*

For the second condenser, we choose $r = \Theta(\log n)$. This gives a condenser with seed $O(\log n)$ that maintains a $(1/\log n)$ -fraction of the initial randomness.

COROLLARY 5.3. *For every n and k such that $k \geq 8 \log^5 n$, there is an explicit $(k, \Omega(k/\log n), 1/\log^2 n)$ -condenser $Con : \{0, 1\}^n \times \{0, 1\}^{O(\log n)} \rightarrow \{0, 1\}^{k/2}$.*

In the remainder of this section we prove Theorem 5.1.

5.1. Getting a block-source. We now implement the idea presented in the introduction, namely, that running the block extraction scheme recursively eventually produces a block-source. In the next definition we recursively run the block extraction scheme. That is, given an n bit string x we use a fresh random seed y of length $O(\log \log n)$ to obtain $x' = B(x, y)$ and continue this process recursively on x' .

DEFINITION 5.4. *Let n, k , and r be parameters such that $k \geq 8 \log^5 n$ and $1 < r \leq \log^2 n$. Let l be a number that will be determined later. Let $t = \log^4 n$ and $\rho = 1/\log^4 n$.*

We first define sequences of numbers n_0, \dots, n_l and k_0, \dots, k_l as follows: $n_i = n/r^i$ and $k_i = k - 2ti$. Let l be the smallest integer such that $n_i < k_i$. We soon show that such an l exists and $l = O(\log_r(n/k))$.

By Lemma 3.3 there exists a universal constant $\gamma > 0$ such that for every i there is a (k_i, ρ, γ) -block extraction scheme $B^i : \{0, 1\}^{n_i} \times \{0, 1\}^{u_i} \rightarrow \{0, 1\}^{n_i/r}$. Furthermore, note that $u_0 \geq u_i$ for every $1 \leq i \leq l$. Let $u = u_0$, and observe that for this choice of parameters, $u = O(\log \log n)$.

For every $0 \leq i \leq l$, we define a function $D_i : \{0, 1\}^n \times (\{0, 1\}^u)^l \rightarrow \{0, 1\}^{n/r^i}$ in the following manner:

- $D_0(x; y_1 \cdots y_l) = x$.
- For $i > 0$, $D_i(x; y_1, \dots, y_l) = B^i(D_{i-1}(x; y_1, \dots, y_l), y_i)$.

It is easy to see that D_i does not depend on y_{i+1}, \dots, y_l , and that for each i , computing D_i takes polynomial time.

Let X be some k -source over n bit strings. Consider the following probability space over $\Omega = \{0, 1\}^n \times (\{0, 1\}^u)^l$. It consists of the random variable X and an independent random variable $Y = (Y_1, \dots, Y_l)$ that is uniformly distributed over $(\{0, 1\}^u)^l$. We also define random variables B_0, \dots, B_l by $B_i = D_i(X, Y)$. Following the intuition in section 1.6, we want to argue that there exists a small l and an $1 \leq i \leq l$ such that (B_i, B_{i-1}) is a block-source. This does not hold. However, we can use the machinery developed in section 4 to show a result with the same flavor.

LEMMA 5.5. *Let $t = \log^4 n$. If $k \geq 8 \log^5 n$, then there exists a partition of $\{0, 1\}^n \times (\{0, 1\}^u)^l$ into $l + 1$ sets: BLK_1, \dots, BLK_l and BAD with the following properties:*

1. $\Pr(BAD) \leq 2l(\rho + 2^{-t})$.
2. (B_i, B_{i-1}) is a (k', t) -block-source in BLK_i , (where $k' \geq \frac{\gamma(k-2lt)}{r}$).
3. $l = O(\log_r(n/k))$.

The remainder of this section is devoted to proving Lemma 5.5. The proof is just a recursive application of Lemma 4.3 and the reader is encouraged to skip it on a first reading.

Proof of Lemma 5.5. For $0 \leq i \leq l$, we recursively define sets $BAD_i, BLK_i, CON_i \subseteq \{0, 1\}^n \times (\{0, 1\}^u)^l$ and a distribution X_i that is defined over strings of length n_i . We define $BAD_0 = BLK_0 = \emptyset$, $CON_0 = \{0, 1\}^n \times (\{0, 1\}^u)^l$, and $X_0 = X$. For $i > 0$, suppose that sets $BAD_{i-1}, BLK_{i-1}, CON_{i-1}$ have already been defined; that the distribution of X_{i-1} is $P_{(B_{i-1}|CON_{i-1})}$; and that X_{i-1} is a (k_{i-1}) -source. (Note that this holds for $i = 1$.) We now recursively define sets BAD_i, BLK_i, CON_i that are a partition of CON_{i-1} and a distribution X_i .

We first apply Lemma 4.3 on the i th application of the block extraction scheme B^i on X_{i-1} and Y_i . It follows that $\{0, 1\}^{n_{i-1}} \times \{0, 1\}^u$ can be partitioned into three sets BAD, BLK, CON as in the lemma.

We “pull these events back to the original probability space.” That is, we want to see these sets as a partition of CON_{i-1} . More precisely, we define

$$BAD_i = \{(x, y_1, \dots, y_l) \in CON_{i-1} : D_{i-1}(x, y_1, \dots, y_l) \in BAD\},$$

$$BLK_i = \{(x, y_1, \dots, y_l) \in CON_{i-1} : D_{i-1}(x, y_1, \dots, y_l) \in BLK\},$$

$$CON_i = \{(x, y_1, \dots, y_l) \in CON_{i-1} : D_{i-1}(x, y_1, \dots, y_l) \in CON\}.$$

Note that this is a partition of CON_{i-1} . Recall that $B_i = D_i(X, Y) = B^i(D_{i-1}(X, Y), Y_i)$. Thus, the distribution $P_{(B_i|CON_i)}$ is exactly the same as $P_{(B^i(X_{i-1}, Y_i)|CON)}$. Similarly, $P_{(B_i|BLK_i)}$ is exactly the same as $P_{(B^i(X_{i-1}, Y_i)|BLK)}$. We conclude that the guarantees of Lemma 4.3 give the following:

1. $\Pr(BAD_i) \leq 2(\rho + 2^{-t})$.
2. (B_i, B_{i-1}) is a $(\frac{\gamma k_{i-1}}{r} - t, t)$ -block-source in BLK_i .
3. B_i is a k_i -source in CON_i .

We now define X_i to be the distribution $P_{(B_i|CON_i)}$ that is over n_i bits. Indeed, we have that X_i is a k_i -source. Thus, we can successfully define sets BAD_i, BLK_i, CON_i such that for each $i > 0$, these sets are a partition of CON_{i-1} and the three properties above hold.

We now show that at some step i , $CON_i = \emptyset$. After i steps, the length of the i th block is $n_i = n/r^i$ and $k_i = k - 2it$. Thus, after $l = \log_r(4n/k)$ steps we have that the i th block is of length at most $k/4$. At this point, $k_l = k - 2lt \geq k - 2\log^5 n \geq k/2$. It follows that $n_l < k_l$ and that there is some $i \leq l$ for which $CON_i = \emptyset$, as otherwise the third item above cannot hold (simply because it is impossible to have a distribution with more entropy than length).

We define $BAD = \cup_{1 \leq i \leq l} BAD_i$. It follows that BLK_1, \dots, BLK_l and BAD are a partition of $\Omega = \{0, 1\}^n \times (\{0, 1\}^l)^u$ and the lemma follows. \square

5.2. Getting a condenser. In the previous section we showed how to get $\ell = O(\log_r(n/k))$ pairs of distributions such that (at least in some sense) one of them is a block-source. Had we been able to construct a single block-source, we could have used the block-source extractor of Corollary 2.6 to get an extractor. At this point, however, we have many candidates (pairs B_i, B_{i-1}). We now run block-source extractors on all pairs (using the same seed). It follows that one of the outputs is close to uniform, and therefore the concatenation of the outputs gives a condenser. We now formalize this intuition.

CONSTRUCTION 5.6. *We use the parameters of Definition 5.4. Let n, k , and r be parameters such that $k \geq 8 \log^5 n$ and $1 < r \leq \log^2 n$. Let $l = \log_r(4n/k)$, $t = \log^4 n$, and $\rho = 1/\log^4 n$. Let $u = O(\log \log n)$ be the seed length for the block extraction scheme as determined in Definition 5.4. Let $k' = \frac{\gamma(k-2lt)}{r}$.*

We define a function $Con : \{0, 1\}^n \times \{0, 1\}^{ul+O(\log n)} \rightarrow \{0, 1\}^{n'}$, where n' is determined later. Given inputs $x \in \{0, 1\}^n$ and $y \in \{0, 1\}^{ul+O(\log n)}$, Con interprets its second argument as l strings $y_1, \dots, y_l \in \{0, 1\}^u$ and an additional string s of length $O(\log n)$. For $0 \leq i \leq l$ it computes $b_i = D_i(x; y_1, \dots, y_l)$ (where D_i is taken from Definition 5.4) and $o_i = BE(b_i, b_{i-1}, s)$ (where BE is the block-source extractor of Corollary 2.6 using output length k'). The final output is (o_1, \dots, o_l) (which makes $n' = lk'$).

We now prove Theorem 5.1.

Proof of Theorem 5.1. Let X be a k -source. For this proof we fix a probability space consisting of independent random variables X, Y , and Z , where $Y = (Y_1, \dots, Y_l)$ is uniformly distributed over $(\{0, 1\}^u)^l$ and $Z = (Z_1, \dots, Z_l)$ is uniformly distributed over $(\{0, 1\}^{k'})^l$. We now define more random variables as a function of the initial random variables. We define random variables B_1, \dots, B_l as before by $B_i = D_i(X, Y)$. We also define random variables O_1, \dots, O_l by $O_i = BE(B_{i-1}, B_i)$. Note that $CON(X, Y) = (O_1, \dots, O_l)$. We also define random variables (R_1, \dots, R_l) over $(\{0, 1\}^{k'})^l$ as follows: Let BLK_1, \dots, BLK_l and BAD be the sets of Lemma 5.5. If $(X; Y_1, \dots, Y_l) \in BAD$, we set $R = Z$. Otherwise, $(X; Y_1, \dots, Y_l)$ belong to a unique BLK_i . In this case we set $R_i = Z_i$ and $R_j = O_j$ for $j \neq i$. Note that R is a k' -source. To complete the proof we now show that (R_1, \dots, R_l) is $(2l(\rho + 2^{-t}) + 2/\log^4 n)$ -close to (O_1, \dots, O_l) . This suffices, as $2l(\rho + 2^{-t}) + 2/\log^4 n \leq 1/\log^2 n$.

By Lemma 5.5 we have that (B_i, B_{i-1}) is close to a block-source in BLK_i . The block-source extractor BE has error $\epsilon' = 1/|B_2| + 1/|B_1|$. Recall that the length of all blocks B_i is at least $k' \geq \log^4 n$. It follows that $\epsilon' < 2/\log^4 n$ and that O_i is ϵ' -close to uniform in BLK_i . We conclude that for every i , (R_1, \dots, R_l) is ϵ' -close to (O_1, \dots, O_l) in BLK_i . This gives that (R_1, \dots, R_l) is ϵ' -close to (O_1, \dots, O_l) in the complement of BAD . By Lemma 5.5 the probability of BAD is at most $2l(\rho + 2^{-t})$. Thus, (R_1, \dots, R_l) is $2l(\rho + 2^{-t}) + \epsilon'$ close to (O_1, \dots, O_l) . \square

Let us compare the entropy rates of the new source and the initial source. The new source has min-entropy k' , which is approximately k , and length approximately

$k \cdot \log \frac{n}{k}$, whereas the initial source had length $n = k \cdot \frac{n}{k}$. Note that $\log(n/k) < n/k$, and thus Con indeed improves the entropy rate and is a $(k, k', 2l(\rho + 2^{-t}) + \epsilon')$ -condenser.

Remark 5.7. Actually, the distribution (O_1, \dots, O_l) is a source of a special kind called a “somewhere random source” by Ta-Shma in [24], where it was shown that extracting randomness from such sources is easier using special extractors, which are called “somewhere random mergers.” At this point we could have used Ta-Shma’s “somewhere random mergers” to extract the randomness from (o_1, \dots, o_l) . Instead, we use different methods, which exploit the fact that l is relatively small, to obtain better results.

6. Constructing extractors. In this section we use the condensers constructed in the previous section to prove the two main theorems (Theorems 1.4 and 1.6).

For Theorem 1.4 we use the condenser of Corollary 5.2 repeatedly (with fresh seeds) to condense the source until we achieve constant entropy rate. (This is guaranteed to happen after no more than $\log^* n$ iterations.) For constant entropy rate, Zuckerman’s extractor [32] (see Table 2) uses the optimal seed length to extract a constant fraction. This procedure loses some randomness in the iteration process and results in an extractor which extracts a subconstant fraction of the initial randomness. We then use the method of [29] to increase this fraction to an arbitrary constant. This informal argument is made precise in the following proof.

Proof of Theorem 1.4. Without loss of generality, we assume that $k \geq 8 \log^5 n$ as the extractor of [7] achieves the required result for $k < 8 \log^5 n$. It is easy to check that, given a (k, k', ϵ) -condenser $Con_1 : \{0, 1\}^n \times \{0, 1\}^d \rightarrow \{0, 1\}^{n'}$ and a (k', k'', ϵ') -condenser $Con_2 : \{0, 1\}^{n'} \times \{0, 1\}^{d'} \rightarrow \{0, 1\}^{n''}$, composing the condensers produces a $(k, k'', \epsilon + \epsilon')$ -condenser $Con : \{0, 1\}^n \times \{0, 1\}^{d+d'} \rightarrow \{0, 1\}^{n''}$.

Let us denote the entropy rate of a source by $R(X) = k/n$ and let $R'(X) = n/k = 1/R(X)$. The condenser of Corollary 5.2 produces a source X' that is close to an $\Omega(k)$ source over $k \log(n/k)$ bits. Thus, $R(X') = \Theta(1/\log(1/R(X)))$ or, in other words, we have that $R'(X') = \Theta(\log(R'(X)))$. We now apply the condenser recursively on X' using a fresh seed. After $\log^* R'(X) \leq \log^* n$ iterations, the entropy rate becomes constant. Once the ratio is constant, Zuckerman’s extractor [32] (see Table 1) can be used to extract a constant fraction (say, half) of the randomness using a fresh seed of length $O(\log n)$ and error $1/\log^2 n$. Overall, we have used at most $\log^* n$ iterations, where in each of them we required a seed of length at most $O(\log n \cdot \log \log n)$ and the final application of Zuckerman’s extractor requires an additional $O(\log n)$ bits. Thus, the strategy described above gives an extractor that uses seed length $O(\log n \cdot \log \log n \cdot \log^* n)$ bits. Recall that our condenser loses a constant fraction of the randomness in every iteration. Thus, after $\log^* n$ iterations we extract only $k/2^{O(\log^* n)}$ random bits from the source and produce an extractor which extracts $1/2^{O(\log^* n)}$ of the initial randomness. To get to a constant fraction we use the method of Wigderson and Zuckerman [29].¹⁰ Implementing the technique of Wigderson and Zuckerman multiplies the seed and error by $2^{O(\log^* n)}$. Thus, the total number of random bits is $\log n \cdot \log \log n \cdot \log^* n \cdot 2^{O(\log^* n)} = O(\log n \cdot (\log \log n)^2)$ as required. Furthermore the final extractor has error smaller than $1/\log n$. \square

In the case of Theorem 1.6, we are shooting for the optimal seed length and cannot afford the condenser of Corollary 5.2 or repeated condensing. Instead we use

¹⁰Wigderson and Zuckerman suggested to repeatedly extract randomness from the source (using fresh seeds) until one extracts the desired fraction. This gives that if $m = k/p$, then m could be increased to $(1 - \alpha)k$ (where α is an arbitrary constant) at the cost of multiplying d by $O(p)$. (An exact formulation of the Wigderson and Zuckerman technique can be found, for example, in [11, 12].)

the condenser of Corollary 5.3, interpreting it as a block extraction scheme. Viewed this way, the condenser extracts a block B of length $k/2$, and therefore the distribution $(B(X, Y), X)$ forms a block-source, since B is too short to “steal” all the randomness from X . (This intuition is formalized in the next lemma.) All that is left is to use the block-source extractor of Corollary 2.6. The precise details follow.

LEMMA 6.1. *Let Con be the condenser of Corollary 5.3. If X is a k -source for $k \geq 8 \log^5 n$, then the distribution $(Con(X, U_{O(\log n)}), X)$ is $O(1/\log^2 n)$ -close to an $(\Omega(k/\log n), \log^4 n)$ -block-source.*

Proof. Fix some n and $k \geq 8 \log^5 n$, and let $Con : \{0, 1\}^n \times \{0, 1\}^{u=O(\log n)} \rightarrow \{0, 1\}^{k/2}$ be the $(k, \Omega(k/\log n), 1/\log^2 n)$ -condenser of Corollary 5.3. For this proof we view Con as a block extraction scheme $B : \{0, 1\}^n \times \{0, 1\}^u \rightarrow \{0, 1\}^{n/r}$ for $r = 2n/k$. It follows that B is a (k, ρ, γ) -block extraction scheme for $\rho = 1/\log^2 n$ and $\gamma = \Omega(r/\log n)$. We remark that, in particular, $\gamma \gg 1$.

We now consider the probability space of section 4. The probability space is over the set $\Omega = \{0, 1\}^n \times \{0, 1\}^u$ and consists of two independent random variables X (the given k -source) and Y (that is uniformly distributed over $\{0, 1\}^u$). We set $t = \log^4 n$ and apply Lemma 4.3 and let BAD, BLK, CON be the sets guaranteed by the lemma. We claim that $CON = \emptyset$.

We make this claim because the lemma guarantees that $(B(X, Y))$ is a $(k - 2t)$ -source in CON . Note that the output length of B is $k/2$, whereas $k - 2t > k/2$ because $k \geq 8 \log^5 n$. Thus, the lemma says that in CON there is a random variable which has min-entropy larger than its length. This statement is true only if $CON = \emptyset$.

Lemma 4.3 also gives that

- $\Pr(BAD) \leq 2(\rho + 2^{-t})$.
- $(B(X, Y), X)$ is a $(\frac{\gamma k}{r} - t, t)$ -block-source in BLK .

Thus, $(B(X, Y), X)$ is $O(\rho + 2^{-t})$ -close to a $(\frac{\gamma k}{r} - t, t)$ -block-source. Using again that $k \geq 8 \log^5 n$, we conclude that the distribution $(Con(X, U_{O(\log n)}), X)$ is $O(1/\log^2 n)$ -close to an $(\Omega(k/\log n), \log^4 n)$ -block-source as required. \square

We now prove Theorem 1.6.

Proof of Theorem 1.6. As in the proof of Theorem 1.4, without loss of generality we can assume that $k \geq 8 \log^5 n$ because the extractor of [7] achieves the required result for $k < 8 \log^5 n$. Given a k -source, we use Lemma 6.1 to get a distribution that is close to a block-source and use the block-source extractor of Corollary 2.6. \square

7. Achieving small error. The statements of Theorems 1.4 and 1.6 are for constant error ϵ . The analysis provided in this paper gives a slightly better result and allows us to replace the requirement that ϵ be a constant with $\epsilon = 1/(\log n)^c$ for any constant c . Still, our technique does not give good dependence of the seed length on the error. We get better dependence on ϵ using the error reduction transformation of [16], which transforms an extractor with large (say, constant) error into an extractor with arbitrary small error, while losing only a little bit in the other parameters. More precisely, after undergoing the transformation, a factor of $O(\log m(\log \log m)^{O(1)} + \log(1/\epsilon))$ is added to d , and the fraction extracted decreases by a constant. The latter loss makes no difference from our point of view since we are only able to extract constant fractions. The first loss is not significant in the case of Theorem 1.4, since the seed size is already larger than the optimal one by a multiplicative $\text{polyloglog}(n)$ factor. However, it completely spoils Theorem 1.6 and makes it inferior to Theorem 1.4. Following is Theorem 1.4 rephrased using the error reduction transformation of [16].

THEOREM 7.1 (*Theorem 1.4 rephrased for nonconstant ϵ*). *For every n, k , and $\epsilon > \exp(\frac{-n}{(\log^* n)^{O(\log^* n)}})$, there are explicit (k, ϵ) -extractors*

$$\text{Ext} : \{0, 1\}^n \times \{0, 1\}^{O(\log n \cdot (\log \log n)^{O(1)} + \log(1/\epsilon))} \rightarrow \{0, 1\}^{(1-\alpha)k},$$

where $\alpha > 0$ is an arbitrary constant.

8. Transforming arbitrary extractors into strong extractors. It is sometimes helpful to have a stronger variant of extractors, called a *strong* extractor. A strong extractor is required to extract randomness “only from the source” and not “from the seed.”

DEFINITION 8.1. *A (k, ϵ) -extractor $\text{Ext} : \{0, 1\}^n \times \{0, 1\}^d \rightarrow \{0, 1\}^m$ is strong if for every k -source X the distribution $(\text{Ext}(X, U_d) \circ U_d)$ (obtained by concatenating the seed to the output of the extractor) is ϵ -close to a U_{m+d} .*

Intuitively, this is helpful since a strong extractor has the property that, for any source, a $1 - \epsilon$ fraction of the seeds extracts randomness from that source. It is interesting to note that the concept of strong extractors preceded that of nonstrong extractors, and the strong version was the one which was defined in the seminal paper [13]. Several extractors constructions (with examples being the constructions of [24, 7] and this paper) are nonstrong or difficult to analyze as strong.

The following theorem shows that every nonstrong extractor can be transformed into a strong one with essentially the same parameters.

THEOREM 8.2. *Any explicit (k, ϵ) -extractor $\text{Ext} : \{0, 1\}^n \times \{0, 1\}^d \rightarrow \{0, 1\}^m$ can be transformed into an explicit strong $(k, O(\sqrt{\epsilon}))$ -extractor $\text{Ext}' : \{0, 1\}^n \times \{0, 1\}^{d+d'} \rightarrow \{0, 1\}^{m-(d+L+1)}$ for $d' = \text{polylog}(d/\epsilon)$ and $L = 2 \log(1/\epsilon) + O(1)$.*

Let us consider the parameters of Ext' compared to that of Ext . The seed length of Ext' is longer than that of Ext by a factor that is only polylogarithmic (for large ϵ). The output length of Ext' is shorter than that of Ext by $d + L + 1$. The loss of d bits is unavoidable, as the output of Ext may contain d bits of randomness from the seed. The additional loss of $L + 1$ bits can sometimes be recovered (at the cost of increasing the seed length). Exact details are given in Remark 8.3.

Remark 8.3. In [17] it was shown that any strong extractor which has seed length d and entropy-loss Δ can be transformed into a strong extractor with seed length $d + O(\Delta)$ and an optimal entropy loss of $2 \log(1/\epsilon) + O(1)$. Thus, if the initial extractor Ext had an entropy loss of Δ , we can use our construction to get an extractor Ext' with the parameters mentioned above, and then use [17] to construct a strong extractor Ext'' with seed length $d'' = d + d' + O(\Delta)$ and optimal entropy-loss. This addition is affordable if Δ is small.

The intuition above also gives a hint for the construction. The output of Ext may contain d bits of randomness which “belong” to the seed. Still, it contains roughly $m - d$ bits which *do not* depend on the seed. Thus, fixing the seed, the output of Ext is a random source of length m which contains roughly $m - d$ random bits. We can now use another extractor to extract this randomness and “dismantle” the correlation between the seed and the output. The extractor we need is one that works well when the source lacks only a very small amount of randomness. Such a construction was given by [4] and improved by [19].

THEOREM 8.4 (see [19]). *There are explicit strong (k, ϵ) -extractors $\text{RVW} : \{0, 1\}^n \times \{0, 1\}^{d'} \rightarrow \{0, 1\}^{k-L}$ for $d' = \text{polylog}((n - k)/\epsilon)$ and $L = 2 \log(1/\epsilon) + O(1)$.*

CONSTRUCTION 8.5. *Given a (k, ϵ) -extractor $\text{Ext} : \{0, 1\}^n \times \{0, 1\}^d \rightarrow \{0, 1\}^m$, we construct Ext' as follows. Let $\text{RVW} : \{0, 1\}^m \times \{0, 1\}^{d' = \text{polylog}(d/\epsilon)} \rightarrow \{0, 1\}^{m-d-1-L}$*

be an $(m - d - 1, \epsilon)$ -extractor guaranteed by Theorem 8.4. Then,

$$\text{Ext}'(x; (y, z)) = \text{RVW}(\text{Ext}(x, y), z).$$

The actual proof that Ext' has the desired properties is slightly more complicated than the above presentation. This is mainly because the above presentation ignores the error of Ext . We now give the formal proof.

Proof of Theorem 8.2. We now describe a probability space for this proof. It consists of three independent random variables: an arbitrary k -source X over n bit strings, a uniformly chosen string Y of length d , and a uniformly chosen string Z of length d' .

Given strings $x \in \{0, 1\}^n$ and $y \in \{0, 1\}^d$ we define the *weight* of (x, y) , denoted $w(x, y)$, in the following way:

$$w(x, y) = \Pr(\text{Ext}(X, Y) = \text{Ext}(x, y)).$$

In other words, this is the weight of the string $\text{Ext}(x, y)$ according to the distribution $\text{Ext}(X, Y)$. We say that a pair (x, y) is heavy if $w(x, y) > 2^{-(m-1)}$. We first claim that only few pairs are heavy.

CLAIM 1. $\Pr((X, Y) \text{ are heavy}) < 2\epsilon$.

Proof. Let V denote the distribution $\text{Ext}(X, Y)$. We now use Lemma 2.2. We have that V is ϵ -close to an m -source (the uniform distribution). Therefore the probability under V of hitting an element v such that $\Pr_V(V = v) > 2^{-(m-1)}$ is bounded by 2ϵ and the claim follows. \square

Call a seed $y \in \{0, 1\}^d$ *bad* if $\Pr((X, y) \text{ is heavy}) > \sqrt{2\epsilon}$. That is, if for many choices of x , the output element is heavy. We now claim that there are few bad seeds.

CLAIM 2. *The fraction of bad seeds is at most $\sqrt{2\epsilon}$.*

Proof. The proof is a Markov argument. If the fraction of bad seeds is more than $\sqrt{2\epsilon}$, then $\Pr((X, Y) \text{ is heavy}) > 2\epsilon$. \square

The following claim shows that running the extractor with a good seed produces a source which lacks very few random bits.

CLAIM 3. *For a good seed y , $\text{Ext}(X, y)$ is $\sqrt{2\epsilon}$ -close to an $(m - d - 1)$ -source.*

Proof. For a good y we know that $\Pr((X, y) \text{ is heavy}) < \sqrt{2\epsilon}$. That is, at least a $1 - \sqrt{2\epsilon}$ fraction of the x 's has $w(x, y) \leq 2^{-(m-1)}$. For such an x ,

$$\begin{aligned} \Pr(\text{Ext}(X, y) = \text{Ext}(x, y)) &= \Pr(\text{Ext}(X, Y) = \text{Ext}(x, y) | Y = y) \\ &\leq \frac{w(x, y)}{2^{-d}} \leq 2^{-(m-d-1)}. \quad \square \end{aligned}$$

We have that Ext' runs RVW on the source $\text{Ext}(X, Y)$ using a fresh seed Z . Using the fact that, for a good seed y , $\text{Ext}(X, y)$ is close to a high entropy source, we derive the following claim.

CLAIM 4. *Given $y \in \{0, 1\}^d$, let D_y denote $(\text{Ext}'(X; (y, Z)) \circ Z)$. For every good y , D_y is $(2\sqrt{\epsilon} + \epsilon)$ -close to uniform.*

Proof. Note that $\text{Ext}'(X; (y, Z)) = \text{RVW}(\text{Ext}(X, y), Z)$. The claim follows immediately from Claim 3 and the fact that RVW is a strong extractor. \square

We now complete the proof of the theorem. Let D denote $(\text{EXT}'(X; (Y, Z)) \circ Z)$. We need to show that $(D \circ Y)$ is $O(\sqrt{\epsilon})$ -close to uniform. Note that $D = D_Y$ (that is, D is a convex combination of the distributions D_y). As the fraction of bad seeds is at most $O(\sqrt{\epsilon})$, it is sufficient to show that for any good seed y , $(D_y \circ Y)$ is $O(\sqrt{\epsilon})$ -close to uniform. Note that as Y is independent of D_y , it is sufficient to show that D_y is $O(\sqrt{\epsilon})$ -close to uniform, which follows from Claim 4. \square

9. Discussion. In a subsequent work, Lu et al. [9] achieve extractors with better parameters than those constructed here. Namely, for constant error $\epsilon > 0$ they achieve a (k, ϵ) -extractor $E : \{0, 1\}^n \times \{0, 1\}^{O(\log n)} \rightarrow \{0, 1\}^{\Omega(k)}$ for every choice of k . Their construction uses some of the ideas of this paper (condensers, win-win analysis) as well as additional ideas.

The next milestone in extractor constructions seems to be achieving seed length $d = O(\log n)$ and output length $m = k + d - O(1)$. (We remark that the difference between output length $\Omega(k)$ and k is significant in some applications of extractors.) This has already been achieved by [25] for small values of k ($k = 2^{\log n / \log \log n}$) in a subsequent work.

Another important goal is to achieve the “correct dependence” of the seed length on ϵ for nonconstant ϵ , namely, to achieve an extractor with seed length $d = O(\log(n/\epsilon))$ and output length (say) $m = \Omega(k)$. We remark that both our approach and the approach of [9] do not give this dependence.

Acknowledgments. We thank Russel Impagliazzo and Amnon Ta-Shma for helpful discussions, and particularly for bringing to our attention their insights on error correcting random sources. We are also grateful to the anonymous referees for helpful comments.

REFERENCES

- [1] M. BLUM, *Independent unbiased coin flips from a correlated biased source—a finite state Markov chain*, *Combinatorica*, 6 (1986), pp. 97–108.
- [2] M. R. CAPALBO, O. REINGOLD, S. P. VADHAN, AND A. WIGDERSON, *Randomness conductors and constant-degree lossless expanders*, in *Proceedings of the 34th Annual ACM Symposium on Theory of Computing*, ACM, New York, 2002, pp. 659–668.
- [3] B. CHOR AND O. GOLDBREICH, *On the power of two-point based sampling*, *J. Complexity*, 5 (1989), pp. 96–106.
- [4] O. GOLDBREICH AND A. WIGDERSON, *Tiny families of functions with random properties: A quality-size trade-off for hashing*, *Random Structures Algorithms*, 11 (1997), pp. 315–343.
- [5] R. IMPAGLIAZZO, *private communication*, 1999.
- [6] R. IMPAGLIAZZO, R. SHALTIEL, AND A. WIGDERSON, *Near-optimal conversion of hardness into pseudo-randomness*, in *Proceedings of the 40th Annual IEEE Symposium on Foundations of Computer Science*, IEEE, Los Alamitos, CA, 1999, pp. 181–190.
- [7] R. IMPAGLIAZZO, R. SHALTIEL, AND A. WIGDERSON, *Extractors and pseudo-random generators with optimal seed length*, in *Proceedings of the 32nd Annual ACM Symposium on Theory of Computing (Portland, OR)*, ACM, New York, 2000, pp. 1–10.
- [8] J. JUSTESEN, *A class of constructive asymptotically good algebraic codes*, *IEEE Trans. Inform. Theory*, 18 (1972), pp. 652–656.
- [9] C. J. LU, O. REINGOLD, S. VADHAN, AND A. WIGDERSON, *Extractors: Optimal up to constant factors*, in *Proceedings of the 35th Annual ACM Symposium on Theory of Computing*, ACM, New York, 2003, pp. 602–611.
- [10] J. NAOR AND M. NAOR, *Small-bias probability spaces: Efficient constructions and applications*, *SIAM J. Comput.*, 22 (1993), pp. 838–856.
- [11] N. NISAN, *Extracting randomness: How and why: A survey*, in *Proceedings of the Eleventh Annual IEEE Conference on Computational Complexity (Philadelphia, PA)*, IEEE, Los Alamitos, CA, 1996, pp. 44–58.
- [12] N. NISAN AND A. TA-SHMA, *Extracting randomness: A survey and new constructions*, *J. Comput. System Sci.*, 58 (1999), pp. 148–173.
- [13] N. NISAN AND D. ZUCKERMAN, *Randomness is linear in space*, *J. Comput. System Sci.*, 52 (1996), pp. 43–52.
- [14] J. RADHAKRISHNAN AND A. TA-SHMA, *Bounds for dispersers, extractors, and depth-two super-concentrators*, *SIAM J. Discrete Math.*, 13 (2000), pp. 2–24.
- [15] R. RAZ AND O. REINGOLD, *On recycling the randomness of states in space bounded computation*, in *Proceedings of the ACM Symposium on Theory of Computing*, ACM, New York, 1999, pp. 159–168.

- [16] R. RAZ, O. REINGOLD, AND S. VADHAN, *Error reduction for extractors*, in Proceedings of the 40th Annual IEEE Symposium on Foundations of Computer Science, IEEE, Los Alamitos, CA, 1999, pp. 191–201.
- [17] R. RAZ, O. REINGOLD, AND S. VADHAN, *Extracting all the randomness and reducing the error in Trevisan's extractors*, J. Comput. System Sci., 65 (2002), pp. 97–128.
- [18] O. REINGOLD, R. SHALTIEL, AND A. WIGDERSON, *Extracting randomness via repeated condensing*, in Proceedings of the 41st Annual IEEE Symposium on Foundations of Computer Science, IEEE, Los Alamitos, CA, 2000, pp. 22–31.
- [19] O. REINGOLD, S. VADHAN, AND A. WIGDERSON, *Entropy waves, the zig-zag graph product, and new constant-degree expanders and extractors*, in Proceedings of the 41st Annual Symposium on Foundations of Computer Science (Redondo Beach, CA), IEEE, Los Alamitos, CA, 2000, pp. 3–13.
- [20] M. SANTHA AND U. V. VAZIRANI, *Generating quasi-random sequences from semi-random sources*, J. Comput. System Sci., 33 (1986), pp. 75–87.
- [21] R. SHALTIEL, *Recent developments in extractors*, Bull. Eur. Assoc. Theor. Comput. Sci., 77 (2002), pp. 67–95.
- [22] R. SHALTIEL AND C. UMANS, *Simple extractors for all min-entropies and a new pseudo-random generator*, in Proceedings of the 42nd Annual IEEE Symposium on Foundations of Computer Science (Las Vegas, NV), 2001, IEEE, Piscataway, NJ, pp. 648–657.
- [23] A. SRINIVASAN AND D. ZUCKERMAN, *Computing with very weak random sources*, SIAM J. Comput., 28 (1999), pp. 1433–1459.
- [24] A. TA-SHMA, *On extracting randomness from weak random sources (extended abstract)*, in Proceedings of the Twenty-Eighth Annual ACM Symposium on the Theory of Computing, (Philadelphia, PA), ACM, New York, 1996, pp. 276–285.
- [25] A. TA-SHMA, C. UMANS, AND D. ZUCKERMAN, *Loss-less condensers, unbalanced expanders, and extractors*, in Proceedings of the 33rd Annual ACM Symposium on Theory of Computing (Hersonissos, Crete, Greece), ACM, New York, 2001, pp. 143–152.
- [26] A. TA-SHMA, D. ZUCKERMAN, AND S. SAFRA, *Extractors from Reed–Muller codes*, in Proceedings of the 42nd Annual IEEE Symposium on Foundations of Computer Science, IEEE, Piscataway, NJ, 2001, pp. 181–190.
- [27] L. TREVISAN, *Extractors and pseudorandom generators*, J. ACM, 48 (2001), pp. 860–879.
- [28] J. VON NEUMANN, *Various techniques used in connection with random digits*, Nat. Bur. Standards Appl. Math. Ser., 12 (1951), pp. 36–38.
- [29] A. WIGDERSON AND D. ZUCKERMAN, *Expanders that beat the eigenvalue bound: Explicit construction and applications*, in Proceedings of the 25th Annual ACM Symposium on the Theory of Computing (San Diego, CA), ACM, New York, 1993, pp. 245–251.
- [30] D. ZUCKERMAN, *General weak random sources*, in Proceedings of the 31st Annual IEEE Symposium on Foundations of Computer Science (St. Louis, MO, IEEE), vol. II, IEEE, Piscataway, NJ, 1990, pp. 534–543.
- [31] D. ZUCKERMAN, *Simulating BPP using a general weak random source*, Algorithmica, 16 (1996), pp. 367–391.
- [32] D. ZUCKERMAN, *Randomness-optimal oblivious sampling*, Random Structures Algorithms, 11 (1997), pp. 345–367.

WORD PROBLEMS AND MEMBERSHIP PROBLEMS ON COMPRESSED WORDS*

MARKUS LOHREY†

Abstract. We consider a compressed form of the word problem for finitely presented monoids, where the input consists of two compressed representations of words over the generators of a monoid \mathcal{M} , and we ask whether these two words represent the same monoid element of \mathcal{M} . Words are compressed using straight-line programs, i.e., context-free grammars that generate exactly one word. For several classes of finitely presented monoids we obtain completeness results for complexity classes in the range from P to EXPSPACE. As a by-product of our results on compressed word problems we obtain a fixed deterministic context-free language with a PSPACE-complete compressed membership problem. The existence of such a language was open so far. Finally, we will investigate the complexity of the compressed membership problem for various circuit complexity classes.

Key words. grammar-based compression, word problems for monoids, context-free languages, complexity

AMS subject classifications. 20F10, 68Q17, 68Q42

DOI. 10.1137/S0097539704445950

1. Introduction. During the last decade, the massive increase in the volume of data has motivated the investigation of algorithms on *compressed data*, e.g., compressed strings, trees, or pictures. The general goal is to develop algorithms that directly work on compressed data without prior decompression. Let us mention here the work on compressed pattern matching; see, e.g., [19, 23, 49, 60].

In this paper we investigate two classes of computational problems on compressed data that have been of central importance in theoretical computer science since its very beginning: the *word problem* and the *membership problem*.

In its most general form, the word problem asks whether two terms over an algebraic structure represent the same element of the structure. Here we restrict ourselves to the word problem for *finitely presented monoids*, i.e., monoids that are given by a finite set of generators and defining relations. In this case the input consists of two finite words over the set of generators, and we ask whether these two words represent the same monoid element. The undecidability results concerning the word problem for finitely presented monoids [47, 56] and finitely presented groups [12, 51] are among the first undecidability results that touched “real mathematics.” Moreover, these negative results motivated a still ongoing investigation of decidable subclasses of word problems and their computational complexity. In particular, monoids that can be presented by *terminating and confluent semi-Thue systems* (i.e., string rewriting systems where every word can be rewritten in a finite number of steps to a unique irreducible word; see [11, 33]) received a lot of attention: these monoids have decidable word problems, and if the restriction to terminating systems is suitably sharpened, then precise complexity bounds can be deduced [10, 41, 42]. All relevant definitions concerning semi-Thue systems and finitely presented monoids are collected in section 3.3.

*Received by the editors August 18, 2004; accepted for publication (in revised form) November 4, 2005; published electronically March 3, 2006.

<http://www.siam.org/journals/sicomp/35-5/44595.html>

†Institut für Formale Methoden der Informatik, Universität Stuttgart, Universitätsstr. 38, 70569 Stuttgart, Germany (lohrey@informatik.uni-stuttgart.de).

In its compressed variant, the input to the word problem for a (finitely presented) monoid consists of two compressed representations of words over the generators. Here we choose *straight-line programs*, or equivalently context-free grammars that generate exactly one word, for compression—this approach is also known as *grammar-based compression*. See section 3.4 for a formal definition of straight-line programs. Recently, straight-line programs turned out to be a very flexible compressed representation of strings. Several other compressed representations, e.g., Lempel–Ziv factorizations [73], can be efficiently converted into straight-line programs and vice versa [55], which implies that most of our complexity results will also hold for Lempel–Ziv factorizations. An algorithmic application of this efficient transformation to and into straight-line programs is given in [23], where the pattern matching problem for Lempel–Ziv compressed texts is solved efficiently via reduction to straight-line programs. Finally, by using straight-line programs for representing inputs, the compressed word problem becomes equivalent to the well-known circuit equivalence problem (a generalization of the circuit evaluation problem), where we ask whether two circuits over a finitely presented monoid \mathcal{M} (i.e., acyclic directed graphs with leaves labeled by generators of \mathcal{M} and internal nodes labeled by the monoid operation) evaluate to the same element of \mathcal{M} . This problem was mainly investigated for finite structures (e.g., finite monoids [7]) and integer circuits [48] so far. From this perspective, the compressed word problem highlights the classical circuit versus formula evaluation problem in the context of finitely presented monoids.

In sections 4–7 we study the complexity of compressed word problems for several subclasses of monoids presented by terminating and confluent semi-Thue systems. For this we will distinguish semi-Thue systems with respect to the syntactical form of the rewriting rules. In sections 4 and 5 we will consider confluent and *2-homogeneous* semi-Thue systems, which are confluent systems where all rules are of the form $w \rightarrow \varepsilon$ with w of length 2. For confluent and 2-homogeneous systems that satisfy a further syntactical restriction (which we call *N-freeness*) we prove that the presented monoid has a polynomial time solvable compressed word problem (Theorem 4.5). For all other confluent and 2-homogeneous systems, the compressed word problem becomes coNP-hard (Corollary 5.9) and is contained in P^{NP} (Theorem 5.1). In section 6 we show that the complexity of the compressed word problem increases to PSPACE-completeness if we allow also rules of the form $u \rightarrow v$, where again u has length 2 but v may have length 0 or 1 (Theorem 6.6)—the resulting semi-Thue systems are called *2-monadic*. The largest class of semi-Thue systems that is considered in this paper consists of confluent and *weight-reducing* systems. It is briefly studied in section 7, where it is shown that the compressed word problem for a monoid which is presented by a confluent and weight-reducing semi-Thue system is in EXPSPACE and is EXPSPACE-hard for some specific system (Theorem 7.1).

As a by-product of our investigation of compressed word problems we obtain several new results concerning compressed membership problems. Here the problem is to decide for a fixed language L (e.g., a regular or context-free language) whether a given straight-line compressed word w belongs to L [55]. Using Theorem 6.6 concerning 2-monadic semi-Thue systems, we show that there exists a fixed deterministic context-free (even deterministic linear) language with a PSPACE-complete compressed membership problem (Corollary 6.7), which solves an open problem from [23, 55]. Corollary 6.7 should be compared with a result from [24], stating that given a straight-line compressed word w and a nondeterministic hierarchical automaton A (see [24] for a precise definition) it is PSPACE-complete to decide whether $w \in L(A)$. It is straightforward to transform a hierarchical automaton in logspace

into an equivalent nondeterministic pushdown automaton of the same size. Corollary 6.7 improves the result of [24] in two aspects: (i) the context-free language in Corollary 6.7 is deterministic and (ii) it is fixed; i.e., it does not belong to the input. Another result related to Corollary 6.7 was recently shown in [50]: it is PSPACE-complete to decide for two given *nonrecursive* context-free grammars G_1 and G_2 whether $L(G_1) \cap L(G_2) \neq \emptyset$. Nonrecursive context-free grammars generate finite languages; in particular, a straight-line program is a nonrecursive context-free grammar. Thus, in comparison to the result of [50], we sharpen the condition on the grammar G_1 (it has to be a straight-line program) but relax the condition on G_2 (it generates an infinite language). One should also note that for the result of [50] it is crucial that both G_1 and G_2 are part of the input.

Compressed membership problems for context-free languages are also interesting in light of recent attempts to use straight-line programs for the compressed representation of control flow traces of procedural programming languages [36, 72]. At a certain level of abstraction, the set of all valid control flow traces of a procedural programming language is a context-free language.

In section 8 we will investigate the complexity of the compressed membership problem for various circuit complexity classes. We show that the levels of the logtime hierarchy [63] correspond in a compressed setting to the levels of the polynomial time hierarchy. This is another instance of a general phenomenon that we observe: in the worst case there is an exponential jump with respect to computational complexity when moving from the (uncompressed) word/membership problem to its compressed variant. This exponential jump is well known also from other work on the complexity of succinct problems [21, 66, 68, 69], where Boolean circuits/formulas are used for the succinct representation of graphs. But whereas for these formalisms general upgrading theorems can be shown, which roughly state that completeness of problem for a complexity class C implies completeness of the compressed variant for the exponentially harder version of C , such an upgrading theorem fails for straight-line programs: The compressed membership problem for a language may be of the same complexity as the language itself (Proposition 8.5). Thus, the relationship between a computational problem and its straight-line compressed variant is quite loose. A similar phenomenon was observed in the context of hierarchical graph descriptions [38], which can be seen as graph generating straight-line programs.

An extended abstract of this paper appeared in [43].

2. Related work. One of the most intensively studied problems on compressed strings is the pattern matching problem; see, e.g., [19, 23, 49, 60]. In these papers, strings are compressed using either a variant of Lempel–Ziv encoding or straight-line programs. Compressed membership problems for straight-line compressed words were investigated for the first time in [23, 55, 59]; see also [62] for a recent survey.

The problem of checking whether for a given input string s and a given integer n there exists a straight-line program of size at most n that generates s is NP-complete [37]. A smallest straight-line program generating a given input string is even hard to approximate up to some constant factor unless $P = NP$ [37]. Practical algorithms for generating a small straight-line program that produces a given input string were proposed and analyzed with respect to their approximation ratios in [16, 37, 61].

Algorithmic problems on compressed data were also investigated for more general structures than only strings. Complexity theoretical investigations of computational problems on compressed graphs can be found in [13, 20, 21, 38, 44, 53, 66, 67, 68, 69]. In [13, 21, 53, 68, 69] (resp., [66]) Boolean circuits (resp., formulas) are used for

compression, in [20, 67] OBBDs are used, and in [38, 44] graphs are represented via hierarchical graph descriptions. The latter formalism can be seen as an adaptation of the idea of grammar-based compression to the context of graphs. Recently, grammar-based compression was also used in the context of XML in order to obtain succinct representations of large XML documents [14, 45, 46, 71]. Here context-free tree grammars are used as a compact representation of XML skeletons.

3. Preliminaries.

3.1. General notation. For a binary relation \rightarrow on some set, we denote by $\xrightarrow{+}$ ($\xrightarrow{*}$) the transitive (reflexive and transitive) closure of \rightarrow . For sets A , B , and C , we write $A = B \uplus C$ if A is the disjoint union of B and C (B or C may be empty). Let Γ be a finite alphabet. The *empty word* over Γ is denoted by ε . Let $s = a_1 a_2 \cdots a_n \in \Gamma^*$ be a word over Γ , where $a_i \in \Gamma$ for $1 \leq i \leq n$. We define $w^{\text{rev}} = a_n a_{n-1} \cdots a_1$. The *alphabet of s* is $\text{alph}(s) = \{a_1, \dots, a_n\}$. The *length* of s is $|s| = n$. Furthermore, for $a \in \Gamma$ we define $|s|_a = |\{i \mid a_i = a\}|$. For $1 \leq i \leq n$ let $s[i] = a_i$ and for $1 \leq i \leq j \leq n$ let $s[i, j] = a_i a_{i+1} \cdots a_j$. If $i > j$, we set $s[i, j] = \varepsilon$. For a subalphabet $\Sigma \subset \Gamma$ we define the projection morphism $\pi_\Sigma : \Gamma^* \rightarrow \Sigma^*$ by $\pi_\Sigma(a) = \varepsilon$ if $a \notin \Sigma$ and $\pi_\Sigma(a) = a$ if $a \in \Sigma$. For a language $L \subseteq \Gamma^*$ we define the language $\text{Pref}(L) = \{w \in \Gamma^* \mid w \text{ is a prefix of some } u \in L\}$. An *involution* $\bar{}$ on Γ is a function $\bar{} : \Gamma \rightarrow \Gamma$ such that $\overline{\overline{a}} = a$ for all $a \in \Gamma$. It may have fixed points; i.e., $\overline{a} = a$ for some $a \in \Gamma$. The involution $\bar{} : \Gamma \rightarrow \Gamma$ can be extended to an involution on Γ^* by setting $\overline{a_1 \cdots a_n} = \overline{a_n} \cdots \overline{a_1}$. By $\overline{\Gamma} = \{\overline{a} \mid a \in \Gamma\}$ we will always denote a disjoint copy of the alphabet Γ . Then we can define an involution $\bar{}$ on $\Delta = \Gamma \cup \overline{\Gamma}$ by setting $\overline{\overline{a}} = a$; this involution will be extended to Δ^* in the above way. A *weight-function* is a homomorphism $f : \Gamma^* \rightarrow \mathbb{N}$ from the free monoid Γ^* with concatenation to the natural numbers with addition such that $f(s) = 0$ if and only if $s = \varepsilon$. Given a linear order \succ on the alphabet Γ , we extend \succ to a linear order on Γ^* , called the *lexicographic extension of \succ* , as follows: $u \succ v$ if either v is a prefix of u or there exist factorizations $u = w a u'$ and $v = w b v'$ with $a, b \in \Gamma$ and $a \succ b$. For two monoids \mathcal{M}_1 and \mathcal{M}_2 we write $\mathcal{M}_1 \cong \mathcal{M}_2$ if \mathcal{M}_1 and \mathcal{M}_2 are isomorphic.

3.2. Complexity theory. We assume that the reader is familiar with standard complexity classes such as P, coNP, PSPACE, and EXPSPACE; see, e.g., [52] for more details. All hardness results in this paper refer to logspace reductions unless some stronger form of reductions is mentioned explicitly. Several times we will use the fact that addition and multiplication of integers with $n^{O(1)}$ many bits can be done in space $O(\log(n))$. In section 8 we will make use of *alternating Turing-machines* with logarithmic running times. An alternating Turing-machine is a nondeterministic Turing-machine, where the set of states is partitioned into existential and universal states [15]. A configuration with a universal (resp., existential) state is accepting if every (resp., some) successor state is accepting. An alternation in a computation of an alternating Turing-machine is a transition from a universal state to an existential state or vice versa. Following [15], we add a random access mechanism to the ordinary Turing-machine model when dealing with sublogarithmic time bounds: There exists a special address tape that contains a binary coded number p . If the machine enters a special query state, then it has random access to the p th symbol of the input. This mechanism allows a Turing-machine to reach every input position in logarithmic time. If the address tape contains a position which is larger than the length of the input, then querying the input yields some distinguished special symbol. With *DLOGTIME* (resp., *ALOGTIME*) we denote the class of languages that can be recognized on a

deterministic (resp., alternating) Turing-machine in time $O(\log(n))$. It is known that ALOGTIME is equal to uniform NC^1 [3], which is the class of all languages that can be recognized by a uniform family of polynomial-size, logarithmic-depth, fan-in 2 Boolean circuits. Functions computable in uniform NC^1 are defined analogously by allowing circuits with more than one output. For the considerations in this paper, it is not necessary to go into the quite technical details of the precise definition of uniformity. We just remark that Ruzzo's U_{E^*} -uniformity of circuit families [58] would be suitable for all purposes in this paper. A language $L \subseteq \{0, 1\}^*$ is NC^1 -many-one reducible to a language $K \subseteq \{0, 1\}^*$, briefly $L \leq_{\text{NC}^1} K$, if there exists a function f which is computable in uniform NC^1 such that for all $x \in \{0, 1\}^*$, $x \in L$ if and only if $f(x) \in K$.

3.3. Semi-Thue systems and finitely presented monoids. Presentations for monoids are the basic concept of this work. In this section we will introduce the necessary definitions. For more details and references see [11, 33].

Let Γ be a finite alphabet. A *semi-Thue system* R over Γ is a finite subset $R \subseteq \Gamma^* \times \Gamma^*$, whose elements are called rules. A rule $(s, t) \in R$ will be also written as $s \rightarrow t$. The pair (Γ, R) is called a *monoid presentation*. The sets $\text{dom}(R)$ of all left-hand sides and $\text{ran}(R)$ of all right-hand sides are defined by $\text{dom}(R) = \{s \mid \exists t : (s, t) \in R\}$ and $\text{ran}(R) = \{t \mid \exists s : (s, t) \in R\}$, respectively. We define two binary relations \rightarrow_R and \leftrightarrow_R on Γ^* as follows:

- $s \rightarrow_R t$ if there exist $u, v \in \Gamma^*$ and $(\ell, r) \in R$ with $s = u\ell v$ and $t = urv$ (the *one-step rewrite relation*).
- $s \leftrightarrow_R t$ if $(s \rightarrow_R t$ or $t \rightarrow_R s)$.

We also write $t_R \leftarrow s$ in case $s \rightarrow_R t$. Let $\text{RED}(R) = \Gamma^* \cdot \text{dom}(R) \cdot \Gamma^*$ be the set of *reducible words* and $\text{IRR}(R) = \Gamma^* \setminus \text{RED}(R)$ be the set of *irreducible words* (with respect to R). The presentation (Γ, R) is *terminating* if there does not exist an infinite chain $s_1 \rightarrow_R s_2 \rightarrow_R s_3 \rightarrow_R \dots$ in Γ^* . The presentation (Γ, R) is *confluent* if for all $s, t, u \in \Gamma^*$ with $t_R \leftarrow^* s \xrightarrow^* u$ there exists $v \in \Gamma^*$ with $t \xrightarrow^* v_R \leftarrow^* u$. It is well known that (Γ, R) is confluent if and only if (Γ, R) is *Church-Rosser*; i.e., for all $s, t \in \Gamma^*$, if $s \leftrightarrow_R^* t$, then $s \xrightarrow^* u_R \leftarrow^* t$ for some $u \in \Gamma^*$. The presentation (Γ, R) is *locally confluent* if for all $s, t, u \in \Gamma^*$ with $t_R \leftarrow s \rightarrow u$ there exists $v \in \Gamma^*$ with $t \xrightarrow^* v_R \leftarrow^* u$. By Newman's lemma, a terminating presentation is confluent if and only if it is locally confluent. Moreover, if (Γ, R) is terminating and confluent, then for every $s \in \Gamma^*$ there exists a unique *normal form* $\text{NF}_R(s) \in \text{IRR}(R)$ such that $s \xrightarrow^* \text{NF}_R(s)$. It is undecidable whether a given presentation is terminating, confluent, or locally confluent, respectively. On the other hand, for a terminating presentation, local confluence (and hence by Newman's lemma also confluence) can be checked using *critical pairs*, which result from overlapping left-hand sides.

The relation \xrightarrow^* is a congruence relation with respect to the concatenation of words; it is called the *Thue-congruence* generated by (Γ, R) . Hence we can define the quotient monoid $\Gamma^* / \xrightarrow^*$, which we denote by $\mathcal{M}(\Gamma, R)$. It is called a *finitely presented monoid*, and we say that $\mathcal{M}(\Gamma, R)$ is the *monoid presented by* (Γ, R) .

A decision problem that is of fundamental importance in the theory of monoid presentations is the word problem. Let (Γ, R) be a fixed presentation. The *word problem* for (Γ, R) is the following decision problem:

INPUT: Two words $s, t \in \Gamma^*$.

QUESTION: Does $s \xrightarrow^* t$ hold; i.e., do s and t represent the same element of the monoid $\mathcal{M}(\Gamma, R)$?

Here the input size is $|s| + |t|$.

If (Γ, R) and (Σ, S) are presentations such that $\mathcal{M}(\Gamma, R) \cong \mathcal{M} \cong \mathcal{M}(\Sigma, S)$, then for every $a \in \Gamma$ there exists a word $w_a \in \Sigma^*$ such that a and w_a represent the same element of \mathcal{M} . If we define the homomorphism $h : \Gamma^* \rightarrow \Sigma^*$ by $h(a) = w_a$, then for all $s, t \in \Gamma^*$ we have $s \stackrel{*}{\leftrightarrow}_R t$ if and only if $h(s) \stackrel{*}{\leftrightarrow}_S h(t)$. Thus, the word problem for (Γ, R) can be reduced to the word problem for (Σ, S) and vice versa. Moreover, this reduction can be realized in deterministic logspace (even in uniform TC^0 [35]). Thus, the decidability and complexity of the word problem do not depend on the chosen presentation. Since we are interested only in decidability and complexity questions for word problems, we may just speak of the word problem for the monoid \mathcal{M} .

It is well known that in case (Γ, R) is a terminating and confluent presentation, then the word problem for $\mathcal{M}(\Gamma, R)$ is decidable: in order to check whether $u \stackrel{*}{\leftrightarrow}_R v$ it suffices to verify whether $\text{NF}_R(u) = \text{NF}_R(v)$. On the other hand, this algorithm does not yield any upper bound on the computational complexity of the word problem [5]. Complexity results on word problems for restricted classes of finitely presented monoids can be found, for instance, in [10, 41, 42].

3.4. Grammar-based compression. Following [55], a *straight-line program* (SLP) (over the terminal alphabet Γ) is a restricted context-free grammar $G = (V, \Gamma, S, P)$ (where V is the set of nonterminals, Γ is the set of terminals, $S \in V$ is the initial nonterminal, and $P \subseteq V \times (V \cup \Gamma)^*$ is the set of productions) such that

- for every $X \in V$ there exists exactly one production of the form $(X, \alpha) \in P$ for $\alpha \in (V \cup \Gamma)^*$, and
- there exists a linear order \prec on the set of nonterminals V such that $X \prec Y$ whenever there exists a production of the form $(X, \alpha) \in P$ with $Y \in \text{alph}(\alpha)$.¹

Clearly, the language generated by the SLP G consists of exactly one word that is denoted by $\text{eval}(G)$. More generally, from every nonterminal $X \in V$ we can generate exactly one word that is denoted by $\text{eval}_G(X)$ (thus $\text{eval}(G) = \text{eval}_G(S)$). We omit the index G if the underlying SLP is clear from the context. We also write $P(G)$ for the set of productions P . The size of G is $|G| = \sum_{(X, \alpha) \in P} |\alpha|$. Note that every SLP can be transformed in polynomial time into an equivalent SLP in *Chomsky normal form*; i.e., all productions have the form (A, a) with $a \in \Gamma$ or (A, BC) with $B, C \in V$.

Example 3.1. Consider the SLP G_7 over the terminal alphabet $\{a, b\}$ that consists of the following productions:

$$X_i \rightarrow X_{i-1}X_{i-2} \text{ for } 3 \leq i \leq 7, \quad X_2 \rightarrow a, \quad X_1 \rightarrow b.$$

X_7 is the start nonterminal. Then $\text{eval}(G_7) = \text{abaababaabaab}$, which is the 7th Fibonacci word. The SLP G_7 is in Chomsky normal form and $|G_7| = 12$.

Sometimes we will also allow exponential expressions of the form A^i for $A \in V$ and $i \in \mathbb{N}$ in the right-hand sides of productions. Here the number i is coded binary. Such an expression can be replaced by a sequence of ordinary productions, where the length of that sequence is bounded polynomially in the length of the binary coding of i . The following lemma collects several simple algorithmic properties of SLPs that will be used several times in this paper.

¹The term “straight-line program” is used for such a context-free grammar, because a production $A \rightarrow \alpha$ corresponds to a definition $A := \alpha$. Thus, the whole context-free grammar can be interpreted as a linear sequence of instructions, i.e., a straight-line program. Usually one also allows instructions in straight-line programs, where the defined variable appears on the right-hand side (e.g., $x := x + 1$). But instructions of this kind can be easily eliminated by introducing additional variables.

LEMMA 3.2. *The following tasks can be easily solved in polynomial time:*

1. *Given an SLP G , calculate $|\text{eval}(G)|$ and $\text{alph}(\text{eval}(G))$.*
2. *Given an SLP G and a number $i \in \{1, \dots, |\text{eval}(G)|\}$, calculate $\text{eval}(G)[i]$.*
3. *Given an SLP G over the terminal alphabet Γ and a homomorphism $\rho : \Gamma^* \rightarrow \Sigma^*$, calculate an SLP H such that $\text{eval}(H) = \rho(\text{eval}(G))$ (this is in fact possible in logspace).*

The proofs of these statements are straightforward. The following result from [30, 54] is much harder to obtain.

LEMMA 3.3 (see [30, 54]). *For two given SLPs G_1 and G_2 , we can decide in polynomial time whether $\text{eval}(G_1) = \text{eval}(G_2)$.*

It is open whether this problem is P-complete. In this work, we will consider the following generalization: Let (Γ, R) be a fixed monoid presentation. The *compressed word problem* for the monoid $\mathcal{M}(\Gamma, R)$ is the following problem:

INPUT: Two SLPs G_1 and G_2 over the terminal alphabet Γ .

QUESTION: Does $\text{eval}(G_1) \overset{*}{\leftrightarrow}_R \text{eval}(G_2)$ hold?

Here the input size is $|G_1| + |G_2|$. Analogously to the uncompressed word problem, the complexity of the compressed word problem does not depend on the chosen presentation (for this, Lemma 3.2, statement 3, can be used). For a fixed language $L \subseteq \Gamma^*$ we will also consider the *compressed membership problem* for the language L , which is the following problem:

INPUT: An SLP G over the terminal alphabet Γ .

QUESTION: Does $\text{eval}(G) \in L$ hold?

We can view the compressed word problem also from another perspective. A *circuit* \mathcal{C} over $\mathcal{M}(\Gamma, R)$ is a finite directed acyclic graph with exactly one node of outdegree 0. The nodes of indegree 0 are labeled with elements from Γ . All nodes of indegree greater than zero are labeled with the multiplication of the monoid $\mathcal{M}(\Gamma, R)$. Such a circuit computes in a natural way an element of $\mathcal{M}(\Gamma, R)$. Then the compressed word problem for $\mathcal{M}(\Gamma, R)$ is equivalent to the question of whether two given circuits over $\mathcal{M}(\Gamma, R)$ compute the same monoid element. This question has been considered in [7] for the case of finite monoids. Clearly, for a finite monoid, the compressed word problem can be solved in polynomial time. In [7], it was shown that for a finite nonsolvable monoid the compressed word problem is P-complete, whereas for every finite solvable monoid the compressed word problem belongs to DET (the class of all problems that are NC¹-reducible to the calculation of an integer determinant [17]) and hence to NC². Due to the tight correspondence between finite monoids and regular languages, every compressed word problem for a finite monoid is equivalent to the compressed membership problem for a specific regular language and vice versa. Thus, [7] gives a complete classification of the complexity of the compressed membership problem for regular languages. Our work can be seen as a first step of an extension of the work from [7] to finitely presented infinite monoids.

Finally, let us remark that most of our complexity results can be transferred to other compression schemes, e.g., Lempel–Ziv 77, briefly LZ77 [73]. If w is a string and G is an SLP of size n with $\text{eval}(G) = w$, then $\text{LZ}(w)$ (the LZ77-compressed representation of w) has size $O(n)$ and can be constructed in polynomial time [55, 61]. On the other hand, if n is the size of $\text{LZ}(w)$, then we can construct in polynomial time an SLP of size $O(n^2 \cdot \log(n))$ that generates w [55]. Thus, if we allow polynomial time reductions, all our hardness results for complexity classes above P also hold if we use LZ77 for compression. Since the transformation from an SLP to the

LZ77-compressed representation might be P-hard, we cannot transfer directly P-hardness results for SLPs to LZ77.

4. Compressed word problems in P. As already mentioned in the previous section, for every finite monoid the compressed word problem is solvable in polynomial time. In this section we will present a class of infinite monoids with polynomial time solvable compressed word problems. This class contains all free groups. We will also prove that the compressed word problem for every free group of rank at least 2 is P-complete.

A presentation (Γ, R) is called 2-homogeneous if, for every $(\ell, r) \in R$, $|\ell| = 2$ and $r = \varepsilon$ [9]. In [42] it was shown that for every 2-homogeneous presentation the (uncompressed) word problem is in logspace. Moreover, the uniform variant of the word problem for 2-homogeneous presentations, where the presentation is part of the input, is complete for symmetric logspace [42].

The following result was shown by Book [9].

PROPOSITION 4.1 (see [9]). *For every 2-homogeneous presentation (Γ, R) there exists a 2-homogeneous and confluent presentation (Σ, S) with $\mathcal{M}(\Gamma, R) \cong \mathcal{M}(\Sigma, S)$.*

For further consideration let us fix a 2-homogeneous presentation (Γ, R) . By Proposition 4.1 we may assume that (Γ, R) is confluent. The following lemma is easy to prove.

LEMMA 4.2. *Let $u, v \in \text{IRR}(R)$. Then there exist factorizations $u = u_1u_2$ and $v = v_1v_2$ such that $\text{NF}_R(uv) = u_1v_2$, $|u_2| = |v_1|$, and $u_2v_1 \xrightarrow{*}_R \varepsilon$.*

The following lemma was shown in [42].

LEMMA 4.3 (see [42]). *There exists a partition $\Gamma = \Sigma_\ell \uplus \Sigma_r \uplus \Delta$ and an involution $\bar{\cdot} : \Delta \rightarrow \Delta$ with $\{(a\bar{a}, \varepsilon) \mid a \in \Delta\} \subseteq R \subseteq \{(a\bar{a}, \varepsilon) \mid a \in \Delta\} \cup \{(ab, \varepsilon) \mid a \in \Sigma_\ell, b \in \Sigma_r\}$.*

We say that (Γ, R) is *N-free* if there do not exist $a, b \in \Sigma_\ell$ and $c, d \in \Sigma_r$ (where Σ_ℓ and Σ_r result from Lemma 4.3) such that $ac, ad, bc \in \text{dom}(R)$ but $bd \notin \text{dom}(R)$. *N-freeness* means that the bipartite graph $(\Sigma_\ell \cup \Sigma_r, \{(a, b) \in \Sigma_\ell \times \Sigma_r \mid (ab, \varepsilon) \in R\})$ does not contain an *N-shaped* induced subgraph; i.e., it is a disjoint union of complete bipartite graphs.

Example 4.4. Let $\Gamma = \{a, \bar{a}, b, \bar{b}\}$ and $R = \{(a\bar{a}, \varepsilon), (\bar{a}a, \varepsilon), (b\bar{b}, \varepsilon), (\bar{b}b, \varepsilon)\}$. Then (Γ, R) is 2-homogeneous, confluent, and *N-free*. In fact, we have $\Delta = \Gamma$ and $\Sigma_\ell = \Sigma_r = \emptyset$. The monoid $\mathcal{M}(\Gamma, R)$ is the free group of rank 2; see also the paragraph before Theorem 4.9. If $\Gamma = \{a, b, c, d\}$ and $R = \{(ac, \varepsilon), (ad, \varepsilon), (bc, \varepsilon)\}$, then (Γ, R) is 2-homogeneous and confluent but not *N-free*. In fact, (Γ, R) is contained in every 2-homogeneous and confluent presentation, which is not *N-free*.

THEOREM 4.5. *If (Γ, R) is 2-homogeneous, confluent, and N-free, then the compressed word problem for $\mathcal{M}(\Gamma, R)$ is in P.*

In the next section we will see that Theorem 4.5 cannot be extended to non-*N-free* presentations unless $\text{P} = \text{NP}$.

The proof of Theorem 4.5 will be presented after introducing *composition systems* [23]—a generalization of SLPs. A composition system $G = (V, \Gamma, S, P)$ is defined analogously to an SLP, but in addition to productions of the form $A \rightarrow \alpha$ ($A \in V$, $\alpha \in (V \cup \Gamma)^*$) it may also contain productions of the form $A \rightarrow B[i, j]$ for $B \in V$ and $i, j \in \mathbb{N}$. For such a production we define $\text{eval}_G(A) = \text{eval}_G(B)[i, j]$; i.e., we select from $\text{eval}_G(B)$ the subword from position i to position j .² We also allow more general

²In [23], a slightly more restricted formalism, where only productions of the form $A \rightarrow B[j, \text{eval}_G(B)]C[1, i]$ are allowed, was introduced. But this definition is easily seen to be equivalent to our formalism.

rules, e.g., $A \rightarrow B[i, j]C[k, \ell]$; of course this does not lead to higher compression rates. As for SLPs we define $\text{eval}(G) = \text{eval}_G(S)$. The following result from [23] generalizes Lemma 3.3.

LEMMA 4.6 (see [23]). *For two given composition systems G_1 and G_2 , we can decide in polynomial time whether $\text{eval}(G_1) = \text{eval}(G_2)$.*

The following result was shown in [29, Ch. 8].

LEMMA 4.7 (see [29]). *A given composition system can be transformed in polynomial time into an SLP that generates the same word.*

Lemma 4.7 leads to an alternative proof of Lemma 4.6: transform the two given composition systems in polynomial time into equivalent SLPs and apply Lemma 3.3. Moreover, Lemma 4.7 implies that all statements from Lemma 3.2 also hold for composition systems.

LEMMA 4.8. *Assume that (Γ, R) is 2-homogeneous, confluent, and N -free. Then the following problem belongs to P :*

INPUT: *Composition systems G_1 and G_2 with $\text{eval}(G_1), \text{eval}(G_2) \in \text{IRR}(R)$, and $|\text{eval}(G_1)| = |\text{eval}(G_2)|$.*

QUESTION: *Does $\text{eval}(G_1)\text{eval}(G_2) \xrightarrow{*}_R \varepsilon$ hold?*

Proof. Let $\Gamma = \Sigma_\ell \uplus \Sigma_r \uplus \Delta$ be the partition resulting from Lemma 4.3 and $\bar{-} : \Delta \rightarrow \Delta$ be the corresponding involution on Δ . Note that $\text{eval}(G_1), \text{eval}(G_2) \in \text{IRR}(R)$ and $\text{eval}(G_1)\text{eval}(G_2) \xrightarrow{*}_R \varepsilon$ implies that $\text{eval}(G_1) \in (\Sigma_\ell \cup \Delta)^*$ and $\text{eval}(G_2) \subseteq (\Sigma_r \cup \Delta)^*$. Thus, we first check in polynomial time whether this is true; if not, we can reject. Next, from G_2 we can easily construct (by reversing productions) a composition system G'_2 with $\text{eval}(G'_2) = \text{eval}(G_2)^{\text{rev}}$. Since (Γ, R) is N -free, we can find partitions $\Sigma_\ell = \uplus_{i=1}^k \Sigma_{\ell,i}$ and $\Sigma_r = \uplus_{i=1}^k \Sigma_{r,i}$ such that

$$R = \{(a\bar{a}, \varepsilon) \mid a \in \Delta\} \cup \bigcup_{i=1}^k \{(ab, \varepsilon) \mid a \in \Sigma_{\ell,i}, b \in \Sigma_{r,i}\}.$$

Let us take new symbols a_1, \dots, a_k and define homomorphisms ρ_1 and ρ_2 by $\rho_1(a) = \rho_2(a) = a_i$ for all $a \in \Sigma_{\ell,i} \cup \Sigma_{r,i}$, $1 \leq i \leq k$, $\rho_1(a) = a$ for all $a \in \Delta$, and $\rho_2(a) = \bar{a}$ for all $a \in \Delta$. From G_1 and G'_2 we can construct in polynomial time composition systems H_1, H_2 such that $\text{eval}(H_1) = \rho_1(\text{eval}(G_1))$ and $\text{eval}(H_2) = \rho_2(\text{eval}(G'_2))$. By construction, we have $\text{eval}(G_1)\text{eval}(G_2) \xrightarrow{*}_R \varepsilon$ if and only if $\text{eval}(H_1) = \text{eval}(H_2)$. The latter identity can be verified in polynomial time by Lemma 4.6. \square

Proof of Theorem 4.5. Let (Γ, R) be a fixed 2-homogeneous, confluent, and N -free presentation. Given SLPs G_1 and G_2 over the terminal alphabet Γ , we have to verify in polynomial time whether $\text{NF}_R(\text{eval}(G_1)) = \text{NF}_R(\text{eval}(G_2))$. By Lemma 4.6, it suffices to prove that given an SLP G in Chomsky normal form over the terminal alphabet Γ , we can construct in polynomial time a composition system H such that $\text{eval}(H) = \text{NF}_R(\text{eval}(G))$. We construct H inductively by adding more and more rules. Initially, we put into $P(H)$ all productions from $P(G)$ of the form $A \rightarrow a$ with $a \in \Gamma$. Now assume that $A \rightarrow BC$ belongs to $P(G)$ and that H already contains enough productions such that $\text{eval}_H(B) = \text{NF}_R(\text{eval}_G(B))$ and $\text{eval}_H(C) = \text{NF}_R(\text{eval}_G(C))$. We first calculate the largest i such that

$$(4.1) \quad \text{eval}_H(B) = u_1v_2, \quad \text{eval}_H(C) = v_1v_2, \quad |u_2| = |v_1| = i, \quad u_2v_1 \xrightarrow{*}_R \varepsilon.$$

Lemma 4.2 implies that $\text{NF}_R(\text{eval}_G(A)) = u_1v_2$. For a given $i \in \mathbb{N}$, we can check condition (4.1) in polynomial time by Lemma 4.8. Since i is bounded exponentially

in the input size, the largest i satisfying (4.1) can be calculated in polynomial time by doing a binary search. For this largest i we add to the current H the production $A \rightarrow B[1, |\text{eval}_H(B)| - i]C[i + 1, |\text{eval}_H(C)|]$. This concludes the proof of Theorem 4.5. \square

For a finite alphabet Γ , the *free group* $F(\Gamma)$ generated by Γ is defined as

$$(4.2) \quad F(\Gamma) = \mathcal{M}(\Gamma \cup \bar{\Gamma}, \{(c\bar{c}, \varepsilon) \mid c \in \Gamma \cup \bar{\Gamma}\}).$$

Recall from section 3.1 that we define an involution on $\Gamma \cup \bar{\Gamma}$ by setting $\bar{\bar{a}} = a$. Clearly, $F(\Gamma)$ is indeed a group. In case $|\Gamma| = n$ we also write F_n instead of $F(\Gamma)$ and call it the *free group of rank n* . It is known that the (uncompressed) word problem for a free group is in logspace [40]. Moreover, the word problem for F_2 is hard for uniform NC¹ [57]. By Theorem 4.5, the compressed word problem for every free group F_n is in P (we have $\Sigma_\ell = \Sigma_r = \emptyset$ for the presentation in (4.2)). This upper bound is also sharp.

THEOREM 4.9. *The compressed word problem for F_2 is P-complete.*

Proof. It suffices to prove P-hardness, which will be done by a reduction from the monotone circuit value problem, i.e., the problem of whether a Boolean circuit consisting of AND and OR gates evaluates to TRUE [25]. We will use the following result, which is proved in [57]: Let $\Gamma = \{a, b\}$ and $x, y \in (\Gamma \cup \bar{\Gamma})^*$ such that $|x| = |y| = k$ and $|x|_a - |x|_{\bar{a}} = |y|_a - |y|_{\bar{a}} = 0$. Then, if we interpret x and y as elements from F_2 , the following holds, where 1 denotes the neutral element of F_2 :

$$\begin{aligned} (x = 1) \vee (y = 1) &\Leftrightarrow \bar{a}^{3k} x a^{3k} y \bar{a}^{3k} \bar{x} a^{3k} \bar{y} = 1, \\ (x = 1) \wedge (y = 1) &\Leftrightarrow \bar{a}^{3k} x a^{3k} y \bar{a}^{3k} x a^{3k} y = 1. \end{aligned}$$

Note that the words on the right of these equivalences have length $16k$ and that the number of a 's minus the number of \bar{a} 's is again 0.

Now let C be a monotone Boolean circuit. Without loss of generality we can assume that C is layered; i.e., the gates of C are partitioned into n layers and a gate in layer $i > 1$ receives its inputs from layer $i - 1$; see, e.g., [28, Problem A.1.6]. Layer 1 contains the input gates and layer n contains the unique output gate. We now construct an SLP $G(C)$ as follows. For every gate z of C , G contains two nonterminals A_z and $A_{\bar{z}}$. The nonterminal A_z will evaluate to a string that represents the 1 of F_2 if and only if gate z of the circuit evaluates to TRUE. The nonterminal $A_{\bar{z}}$ evaluates to the inverse of $\text{eval}_{G(C)}(A_z)$ in F_2 . Moreover, we will have $|\text{eval}_{G(C)}(A_z)| = |\text{eval}_{G(C)}(A_{\bar{z}})| = 2 \cdot 16^{i-1}$ if z is located in the i th layer of the circuit ($1 \leq i \leq n$).

For every input gate x in layer 1 we introduce the productions

$$\begin{aligned} A_x &\rightarrow \begin{cases} a\bar{a} & \text{if input gate } x \text{ is TRUE,} \\ b^2 & \text{if input gate } x \text{ is FALSE,} \end{cases} \\ A_{\bar{x}} &\rightarrow \begin{cases} a\bar{a} & \text{if input gate } x \text{ is TRUE,} \\ \bar{b}^2 & \text{if input gate } x \text{ is FALSE.} \end{cases} \end{aligned}$$

If z is an OR gate in the i th layer ($i \geq 2$) with input gates x and y from the $(i - 1)$ th layer, then the productions for A_z and $A_{\bar{z}}$ are

$$\begin{aligned} A_z &\rightarrow \bar{a}^{6 \cdot 16^{i-2}} A_x a^{6 \cdot 16^{i-2}} A_y \bar{a}^{6 \cdot 16^{i-2}} A_{\bar{x}} a^{6 \cdot 16^{i-2}} A_{\bar{y}} \text{ and} \\ A_{\bar{z}} &\rightarrow A_y \bar{a}^{6 \cdot 16^{i-2}} A_x a^{6 \cdot 16^{i-2}} A_{\bar{y}} \bar{a}^{6 \cdot 16^{i-2}} A_{\bar{x}} a^{6 \cdot 16^{i-2}}. \end{aligned}$$

Note that the binary codings of the exponents $6 \cdot 16^{i-2}$ have polynomial length, and hence each of the above productions can be replaced by a sequence of ordinary productions. Moreover, if $|\text{eval}(A_u)| = 2 \cdot 16^{i-2}$ for $u \in \{x, \bar{x}, y, \bar{y}\}$ (which is true if x

and y are located in the first layer, i.e., $i = 2$), then $|\text{eval}(A_z)| = |\text{eval}(A_{\bar{z}})| = 2 \cdot 16^{i-1}$. If z is an AND gate in the i th layer ($i \geq 2$) with input gates x and y , then the productions for A_z and $A_{\bar{z}}$ are

$$A_z \rightarrow \bar{a}^{6 \cdot 16^{i-2}} A_x a^{6 \cdot 16^{i-2}} A_y \bar{a}^{6 \cdot 16^{i-2}} A_x a^{6 \cdot 16^{i-2}} A_y \text{ and}$$

$$A_{\bar{z}} \rightarrow A_y \bar{a}^{6 \cdot 16^{i-2}} A_{\bar{x}} a^{6 \cdot 16^{i-2}} A_{\bar{y}} \bar{a}^{6 \cdot 16^{i-2}} A_{\bar{x}} a^{6 \cdot 16^{i-2}}.$$

Once again, these productions can be replaced by a sequence of ordinary productions. Let o be the unique output gate of the circuit C . Then, by the result from [57], the circuit C evaluates to TRUE if and only if $\text{eval}_{G(C)}(A_o) = 1$ in F_2 . \square

5. Compressed word problems between P and PSPACE. In this section we will consider 2-homogeneous and confluent presentations that are not necessarily N -free. Recall that P^{NP} is the class of all languages that can be accepted by a deterministic polynomial time machine that has additional access to an NP-oracle [64]. P^{NP} is also denoted by Δ_2^P in the literature; it is contained in the second level of the polynomial time hierarchy and hence in PSPACE. Several complete problems for P^{NP} can be found in [34].

THEOREM 5.1. *If (Γ, R) is 2-homogeneous and confluent (but not necessarily N -free), then the compressed word problem for $\mathcal{M}(\Gamma, R)$ is in P^{NP} .*

Proof. The key observation is that for a 2-homogeneous and confluent (but not necessarily N -free) presentation (Γ, R) the problem from Lemma 4.8 is in coNP ; i.e., the complementary condition is in NP: If $\text{eval}(G_1), \text{eval}(G_2) \in \text{IRR}(R)$ and $|\text{eval}(G_1)| = |\text{eval}(G_2)|$, then

$$\text{eval}(G_1)\text{eval}(G_2) \xrightarrow{*}_R \varepsilon \text{ does not hold}$$

if and only if there exists $1 \leq i \leq |\text{eval}(G_1)|$ with

$$\text{eval}(G_1)[i] \text{eval}(G_2)[|\text{eval}(G_2)| - i + 1] \notin \text{dom}(R).$$

For a given i , the latter condition can be checked in polynomial time. Now the decision procedure from the proof of Theorem 4.5 in section 4 gives us a P^{coNP} -, i.e., P^{NP} -,algorithm in the present situation. \square

By the next result, coNP -hardness can be obtained for every 2-homogeneous presentations that is not N -free.

THEOREM 5.2. *Let $\Gamma = \{a, b, c, d\}$ and $R = \{(ac, \varepsilon), (ad, \varepsilon), (bc, \varepsilon)\}$. The compressed word problem for $\mathcal{M}(\Gamma, R)$ is coNP -hard.*

Proof. The following problem is the complementary problem to SUBSETSUM [22, Problem SP13], and hence is coNP -complete:

INPUT: Binary coded integers $w_1, \dots, w_n, t \geq 0$.

QUESTION: For all $x_1, \dots, x_n \in \{0, 1\}$, does $\sum_{i=1}^n x_i \cdot w_i \neq t$ hold?

Let us fix binary coded integers $w_1, \dots, w_n, t \geq 0$. Let $\bar{w}_k = (w_1, \dots, w_k)$, and $\bar{w} = \bar{w}_n$. Let $\bar{1}_k = (1, \dots, 1)$ be the k -dimensional vector with all entries equal to 1. For vectors $\bar{x} = (x_1, \dots, x_m)$ and $\bar{y} = (y_1, \dots, y_m)$ we define $\bar{x} \cdot \bar{y} = x_1 y_1 + \dots + x_m y_m$. Finally, let $s_k = \bar{1}_k \cdot \bar{w}_k = w_1 + \dots + w_k$, and $s = s_n = w_1 + \dots + w_n$.

We construct two SLPs G_1 and G_2 over the terminal alphabets $\{a, b\}$ and $\{c, d\}$, respectively, such that $\text{eval}(G_1)\text{eval}(G_2) \xrightarrow{*}_R \varepsilon$ (i.e., $\text{eval}(G_1)\text{eval}(G_2) \xleftrightarrow{*}_R \varepsilon$ since R

is confluent and hence Church–Rosser) if and only if $\bar{x} \cdot \bar{w} \neq t$ for all $\bar{x} \in \{0, 1\}^n$. This will prove the theorem. First, let us construct G_1 :

$$\begin{aligned} S_1 &\rightarrow ba^{s+w_1}b, \\ S_{k+1} &\rightarrow S_k a^{s-s_k+w_{k+1}}S_k. \end{aligned}$$

Let S_n be the start nonterminal of G_1 .

*Claim.*³

$$\text{eval}_{G_1}(S_k) = \left(\prod_{\bar{x} \in \{0,1\}^k \setminus \{\bar{1}_k\}} a^{\bar{x} \cdot \bar{w}_k} b a^{s - \bar{x} \cdot \bar{w}_k} \right) a^{s_k} b.$$

We prove the claim by induction on k . The case $k = 1$ is clear, since $\text{eval}_{G_1}(S_1) = ba^{s+w_1}b = ba^s a^{s_1} b = a^0 b a^{s-0} a^{s_1} b$. For $k + 1 \leq n$ we obtain the following:

$$\begin{aligned} &\left(\prod_{\bar{x} \in \{0,1\}^{k+1} \setminus \{\bar{1}_{k+1}\}} a^{\bar{x} \cdot \bar{w}_{k+1}} b a^{s - \bar{x} \cdot \bar{w}_{k+1}} \right) a^{s_{k+1}} b \\ &= \underbrace{\left(\prod_{\bar{x} \in \{0,1\}^k} a^{\bar{x} \cdot \bar{w}_k} b a^{s - \bar{x} \cdot \bar{w}_k} \right)}_{\text{eval}(S_k) a^{s-s_k}} \underbrace{\left(\prod_{\bar{x} \in \{0,1\}^k \setminus \{\bar{1}_k\}} a^{\bar{x} \cdot \bar{w}_k + w_{k+1}} b a^{s - \bar{x} \cdot \bar{w}_k - w_{k+1}} \right)}_{a^{w_{k+1}} \text{eval}(S_k)} a^{w_{k+1}} a^{s_k} b \\ &= \text{eval}_{G_1}(S_k) a^{s-s_k+w_{k+1}} \text{eval}_{G_1}(S_k) = \text{eval}_{G_1}(S_{k+1}), \end{aligned}$$

which proves the claim.

For $k = n$ we get

$$\text{eval}(G_1) = \text{eval}_{G_1}(S_n) = \prod_{\bar{x} \in \{0,1\}^n} a^{\bar{x} \cdot \bar{w}} b a^{s - \bar{x} \cdot \bar{w}}.$$

Now let G_2 be an SLP such that $\text{eval}(G_2) = (c^{s-t} d c^t)^{2^n}$, which is easy to construct. Let us give an example before we continue with the proof.

Example 5.3. Assume that $w_1 = 2$, $w_2 = 5$, $w_3 = 8$, and $t = 9$. Thus, $s = 2 + 5 + 8 = 15$ and

$$\begin{aligned} \text{eval}(G_1) &= ba^{15} a^2 b a^{13} a^5 b a^{10} a^7 b a^8 a^8 b a^7 a^{10} b a^5 a^{13} b a^2 a^{15} b \\ &= ba^{17} b a^{18} b a^{17} b a^{16} b a^{17} b a^{18} b a^{17} b \\ \text{eval}(G_2) &= (c^6 d c^9)^8 = c^6 d c^{15} d c^{15} d c^{15} d c^{15} d c^{15} d c^{15} d c^9. \end{aligned}$$

For this example, we have $\text{eval}(G_1) \text{eval}(G_2) \xrightarrow{*}_R \varepsilon$, because, while reducing the word $\text{eval}(G_1) \text{eval}(G_2)$, in the middle of the current word the factor bd (which cannot be replaced by ε) will never occur.

Note that $\text{eval}(G_1) \in \{a, b\}^*$, $\text{eval}(G_2) \in \{c, d\}^*$, and $|\text{eval}(G_1)| = |\text{eval}(G_2)| = 2^n \cdot (s + 1)$. Thus, since bd is the only factor from $\{ac, ad, bc, bd\}$ that cannot be

³The \prod -expression on the right-hand side of this production denotes the concatenation of the corresponding words, where the order of concatenation is given by the lexicographic order on the set of vectors $\{0, 1\}^k$, where the last position has the highest significance.

rewritten to ε , we have $\text{eval}(G_1)\text{eval}(G_2) \xrightarrow{*}_R \varepsilon$ if and only if a b does not meet a d in the unique R -derivation that starts from $\text{eval}(G_1)\text{eval}(G_2)$. Hence, by construction of G_1 and G_2 , we have $\text{eval}(G_1)\text{eval}(G_2) \xrightarrow{*}_R \varepsilon$ if and only if $\bar{x} \cdot \bar{w} \neq t$ for all $\bar{x} \in \{0, 1\}^n$. This concludes the proof. \square

The precise complexity of the compressed word problem for a 2-homogeneous but not N -free presentation remains open; it is located somewhere between coNP and P^{NP} . On the other hand, by the previous proof, it is already coNP -hard to decide whether $\text{eval}(G) \xleftrightarrow{*}_R \varepsilon$ for a given SLP G , in case R is 2-homogeneous and confluent but not N -free. For this restricted variant of the compressed word problem we can also prove an upper bound of coNP .

THEOREM 5.4. *For every 2-homogeneous and confluent (but not necessarily N -free) presentation (Γ, R) , the following problem belongs to coNP :*

INPUT: An SLP over the terminal alphabet Γ .

QUESTION: Does $\text{eval}(G) \xleftrightarrow{}_R \varepsilon$ (i.e., $\text{eval}(G) \xrightarrow{*}_R \varepsilon$) hold?*

For the proof of Theorem 5.4 we first introduce some notation.

Let us fix a 2-homogeneous and confluent (but not necessarily N -free) presentation (Γ, R) for the rest of this section. Recall that by Lemma 4.3, there exist a partition $\Gamma = \Sigma_\ell \uplus \Sigma_r \uplus \Delta$ and an involution $\bar{\cdot} : \Delta \rightarrow \Delta$ such that $\{(a\bar{a}, \varepsilon) \mid a \in \Delta\} \subseteq R \subseteq \{(a\bar{a}, \varepsilon) \mid a \in \Delta\} \cup \{(ab, \varepsilon) \mid a \in \Sigma_\ell, b \in \Sigma_r\}$. Let $S = \{(a\bar{a}, \varepsilon) \mid a \in \Delta\} \subseteq R$, which is also 2-homogeneous and confluent but in addition N -free. Thus, by Theorem 4.5, the compressed membership problem for the language $\{w \in \Delta^* \mid w \xrightarrow{*}_S \varepsilon\}$ can be solved in polynomial time.

Let us take two bracket symbols “ \langle ” and “ \rangle ” and define the morphism $\rho : \Gamma^* \rightarrow \{\langle, \rangle\}^*$ by $\rho(a) = \langle$ for $a \in \Sigma_\ell$, $\rho(b) = \rangle$ for $b \in \Sigma_r$ and $\rho(c) = \varepsilon$ for $c \in \Delta$. Let D_1 be the set of all well-bracketed words over $\{\langle, \rangle\}$; i.e., D_1 is the Dyck language over one bracket pair. If P is the semi-Thue system that contains the single rule $\langle \rangle \rightarrow \varepsilon$, then $D_1 = \{w \in \{\langle, \rangle\}^* \mid w \xrightarrow{*}_P \varepsilon\}$. Since P is 2-homogeneous, confluent, and N -free, Theorem 4.5 implies that the compressed membership problem for D_1 can be solved in polynomial time.

Now assume that $w \in \Gamma^*$ is a word such that $\rho(w) \in D_1$. We say that two positions $i, j \in \{1, \dots, |w|\}$ are *corresponding brackets*, briefly $\text{match}(i) = j$, if $w[i] \in \Sigma_\ell$, $w[j] \in \Sigma_r$, $i < j$, $\rho(w[i, j]) \in D_1$, and $\rho(w[i, k]) \notin D_1$ for all k with $i < k < j$.

Example 5.5. Let $\Gamma = \{a, b, c, d, x, y, \bar{x}, \bar{y}\}$ and

$$R = \{(ac, \varepsilon), (ad, \varepsilon), (bc, \varepsilon), (x\bar{x}, \varepsilon), (\bar{x}x, \varepsilon), (y\bar{y}, \varepsilon), (\bar{y}y, \varepsilon)\}.$$

Thus, $\Delta = \{x, y, \bar{x}, \bar{y}\}$, $\Sigma_\ell = \{a, b\}$, and $\Sigma_r = \{c, d\}$. Consider the word $w = x a \bar{y} y c y \bar{x} b a x \bar{x} d x c \bar{y} \bar{x}$. Then $\rho(w) = \langle \rangle \langle \langle \rangle \rangle \in D_1$ and, for instance, $\text{match}(8) = 14$.

LEMMA 5.6. *The following problem can be solved in polynomial time:*

INPUT: An SLP G over the terminal alphabet Γ such that $\rho(\text{eval}(G)) \in D_1$ and a position $1 \leq i \leq |\text{eval}(G)|$ such that $\text{eval}(G)[i] \in \Sigma_\ell$.

OUTPUT: The unique position $j = \text{match}(i)$ in $\text{eval}(G)$.

Proof. In a first step we will reduce the problem to the alphabet $\{\langle, \rangle\}$. Let $1 \leq i \leq |\text{eval}(G)|$ such that $\text{eval}(G)[i] \in \Sigma_\ell$. First, we calculate in polynomial time the unique number k such that i is the position of the k th symbol from $\Sigma_\ell \cup \Sigma_r$ in $\text{eval}(G)$. Formally, $k = |\pi_{\Sigma_\ell \cup \Sigma_r}(\text{eval}(G)[1, i])|$ (by Lemma 4.7 we can calculate in polynomial time an SLP H that generates $\text{eval}(G)[1, i]$; then $|\pi_{\Sigma_\ell \cup \Sigma_r}(\text{eval}(H))|$ can be calculated in polynomial time using Lemma 3.2). Now assume for a moment that we can calculate the position ℓ of the bracket “ \rangle ” that corresponds to the bracket “ \langle ” at position k in $\rho(\text{eval}(G))$ —we also call this position $\text{match}(k)$. Then we just calculate

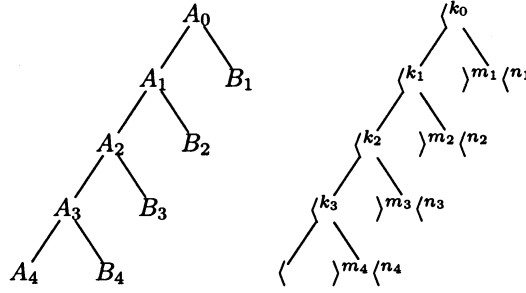


FIG. 5.1.

the position j of the ℓ th symbol from $\Sigma_\ell \cup \Sigma_r$ in $\text{eval}(G)$ and have $\text{match}(i) = j$ in $\text{eval}(G)$. Formally, j is the unique number with $\ell = |\pi_{\Sigma_\ell \cup \Sigma_r}(\text{eval}(G)[1, j])|$ and $\text{eval}(G)[j] \in \Sigma_\ell \cup \Sigma_r$. In order to calculate j from ℓ in polynomial time, we calculate bottom-up $|\pi_{\Sigma_\ell \cup \Sigma_r}(\text{eval}_G(A))|$ as well as $|\text{eval}_G(A)|$ for every nonterminal A of the SLP G . Then we can walk top-down in G in order to calculate j from ℓ .

Thus, we may assume that G is an SLP with $\text{eval}(G) \in D_1$ and $1 \leq i \leq |\text{eval}(G)|$ is a position with $\text{eval}(G)[i] = \langle$. We compute in polynomial time the unique matching position $\text{match}(i)$ in $\text{eval}(G)$. Consider the language

$$K = \text{Pref}(D_1) \setminus [(D_1 \setminus \{\varepsilon\}) \cdot \text{Pref}(D_1)];$$

i.e., K is the set of all prefixes of words from D_1 that do not contain a nonempty prefix from D_1 . Then

$$\text{match}(i) = \max\{j > i \mid \text{eval}(G)[i, j] \in K\} + 1.$$

Since K is prefix-closed, it suffices to show that the compressed membership problem for the language K is solvable in polynomial time, because then we can find the largest $j > i$ with $\text{eval}(G)[i, j] \in K$ using a binary search.

Thus, let H be an SLP in Chomsky normal form. We want to check whether $\text{eval}(H) \in K = \text{Pref}(D_1) \setminus [(D_1 \setminus \{\varepsilon\}) \cdot \text{Pref}(D_1)]$. Recall that the semi-Thue system P consists of the single rule $\langle \rangle \rightarrow \varepsilon$. Clearly, for a word $w \in \{\langle, \rangle\}^*$, we have $\text{NF}_P(w) \in \rangle^* \langle^*$. We represent a word $\rangle^n \langle^m$ by the binary coding of n and m . Using this representation, we can calculate bottom-up in polynomial time for every nonterminal A of H the normal form $\text{NF}_P(\text{eval}_H(A)) \in \rangle^* \langle^*$. Clearly, for every word $w \in \{\langle, \rangle\}^*$, $w \in \text{Pref}(D_1)$ if and only if $\text{NF}_P(w) \in \langle^*$. Using this, we can first check whether $\text{eval}(H) = \text{eval}_H(S) \in \text{Pref}(D_1)$. If this is not the case, we reject. Thus, assume that $\text{eval}(H) \in \text{Pref}(D_1)$. We have to check in polynomial time whether or not $\text{eval}(H) \in (D_1 \setminus \{\varepsilon\}) \cdot \text{Pref}(D_1)$.

Consider the unique path of nonterminals $S = A_0, A_1, \dots, A_m$ such that $A_m \rightarrow \langle$ and for all $i \leq 0 < m$, $A_i \rightarrow A_{i+1}B_{i+1}$ are productions of H . This path is shown in Figure 5.1 (left) for $m = 4$. Since every prefix of $\text{eval}(H)$ also belongs to $\text{Pref}(D_1)$, we have $\text{NF}_P(\text{eval}_H(A_i)) = \langle^{k_i}$ for some numbers $k_i \geq 0$. Assume that $\text{NF}_P(\text{eval}_H(B_i)) = \rangle^{m_i} \langle^{n_i}$ for $m_i, n_i \geq 0$; see the right tree in Figure 5.1. We claim that $\text{eval}(H) \in (D_1 \setminus \{\varepsilon\}) \cdot \text{Pref}(D_1)$ if and only if there exists $1 \leq i \leq m$ such that $k_i \leq m_i$, which can be checked in polynomial time. If $k_i \leq m_i$ for some $1 \leq i \leq m$, i.e., $\text{NF}_P(\text{eval}_H(B_i)) = \rangle^{k_i} \langle^\ell \langle^{n_i}$ for some ℓ , then there exists a prefix u of $\text{eval}_H(B_i)$ such that $\text{NF}_P(u) = \rangle^{k_i}$ and thus $\text{eval}_H(A_i)u \in D_1$ (we may have $u = \varepsilon$ in case $k_i = 0$, i.e.,

$\text{eval}_H(A_i) \in D_1$). The word $\text{eval}_H(A_i)u$ is a nonempty prefix of $\text{eval}(H)$ that belongs to D_1 . On the other hand, if there exists a nonempty prefix $v \in D_1$ of $\text{eval}(H)$, then let i be maximal such that v is a prefix of $\text{eval}_H(A_i)$. Clearly, $i < m$, since $\text{eval}_H(A_m) = \langle$. Thus, $v = \text{eval}_H(A_{i+1})u$ for some prefix u of $\text{eval}_H(B_{i+1})$. Since $\text{NF}_P(\text{eval}_H(A_{i+1})) = \langle^{k_{i+1}}$ and $\text{eval}_H(A_{i+1})u \in D_1$, it follows that $\text{NF}_P(u) = \rangle^{k_{i+1}}$. Since $\text{NF}_P(\text{eval}_H(B_{i+1})) = \rangle^{m_{i+1}}\langle^{n_{i+1}}$ and u is a prefix of $\text{eval}_H(B_{i+1})$, we obtain $m_{i+1} \geq k_{i+1}$. This proves the lemma. \square

Let us again take a word $w \in \Gamma^*$ such that $\rho(w) \in D_1$. Then we can factorize w uniquely as $w = s_0 w[i_1, j_1] s_1 \cdots w[i_n, j_n] s_n$, where $n \geq 0$, $\text{match}(i_k) = j_k$ for all $k \in \{1, \dots, n\}$, and $s_k \in \Delta^*$ for all $k \in \{0, \dots, n\}$. We define $\mathcal{F}(w) = s_0 s_1 \cdots s_n \in \Delta^*$.

Example 5.7. Take Γ, R , and the word $w \in \Gamma^*$ from Example 5.5. Then we have $\mathcal{F}(w) = x y \bar{x} \bar{y} \bar{x}$.

LEMMA 5.8. *The following problem can be solved in polynomial time:*

INPUT: An SLP G over the terminal alphabet Γ such that $\rho(\text{eval}(G)) \in D_1$ and two positions i and j such that $\text{match}(i) = j$ in $\text{eval}(G)$.

OUTPUT: An SLP that generates $\mathcal{F}(\text{eval}(G)[i + 1, j - 1])$.

Proof. Let $\Theta = \Delta \cup \{\langle, \rangle\}$ and consider the infinite semi-Thue system

$$T = \{\langle w \rangle \rightarrow \varepsilon \mid w \in \Delta^*\}$$

over the alphabet Θ . This system is clearly terminating and confluent (T has no overlapping left-hand sides); hence every word $w \in \Theta^*$ has a unique normal form $\text{NF}_T(w)$. Note that $\text{IRR}(T) = (\Delta^*)\Delta^*(\Delta^*\langle\Delta^*)^*$. Let $\mu : \Gamma \rightarrow \Theta$ be the morphism with $\mu(a) = \langle$ for all $a \in \Sigma_\ell$, $\mu(a) = \rangle$ for all $a \in \Sigma_r$, and $\mu(a) = a$ for all $a \in \Delta$.

In a first step we construct in polynomial time an SLP G' such that $\text{eval}(G') = \mu(\text{eval}(G)[i + 1, j - 1])$. Then $\text{NF}_T(\text{eval}(G')) = \mathcal{F}(\text{eval}(G)[i + 1, j - 1])$. Hence, it suffices to calculate a composition system H that generates $\text{NF}_T(\text{eval}(G))$ for a given SLP G with $\text{eval}(G) \in \Theta^*$; this composition system can be transformed in polynomial time into an equivalent SLP by Lemma 4.7. We construct H analogously to the proof of Theorem 4.5. Assume that $A \rightarrow BC$ is a production from G and assume that H already contains enough rules such that $\text{eval}_H(X) = \text{NF}_T(\text{eval}_G(X)) =: w_X \in (\Delta^*)\Delta^*(\Delta^*\langle\Delta^*)^*$ for $X \in \{B, C\}$. We then calculate the numbers $n_B = |w_B|_\langle$ and $n_C = |w_C|_\rangle$, i.e., the number of opening (closing) brackets in w_B (w_C). Assume that $n_C \geq n_B$; the other case is analogous. Then we calculate the position i_B of the n_B th opening bracket \langle in w_B as well as the position i_C of the n_C th closing bracket \rangle in w_C . It follows that $\text{NF}_T(\text{eval}_G(A)) = w_B[1, i_B - 1]w_C[i_C + 1, |w_C|]$. Thus, we add to the current H the production $A \rightarrow B[1, i_B - 1]C[i_C + 1, |w_C|]$. \square

Proof of Theorem 5.4. We use all notation from the previous discussion. The following statement was shown in [42]: For $w \in \Gamma^*$ it holds that $w \stackrel{*}{\leftrightarrow}_R \varepsilon$ if and only if $\rho(w) \in D_1$, $\mathcal{F}(w) \stackrel{*}{\rightarrow}_S \varepsilon$, and for all $i, j \in \{1, \dots, |w|\}$ with $\text{match}(i) = j$ it holds that $w[i]w[j] \in \text{dom}(R)$ and $\mathcal{F}(w[i + 1, j - 1]) \stackrel{*}{\rightarrow}_S \varepsilon$.

This leads to the following NP-algorithm for testing $\text{eval}(G) \stackrel{*}{\leftrightarrow}_R \varepsilon$ for a given SLP G , and hence to a coNP-algorithm for testing $\text{eval}(G) \stackrel{*}{\leftrightarrow}_R \varepsilon$:

1. Produce an SLP G' such that $\text{eval}(G') = \rho(\text{eval}(G))$ and check whether $\text{eval}(G') \in D_1$. This is possible in polynomial time by Theorem 4.5. If $\text{eval}(G') \notin D_1$, then accept; otherwise continue.
2. Guess a position $1 \leq i \leq |\text{eval}(G)|$ (this is the only nondeterministic step) such that $\text{eval}(G)[i] \in \Sigma_\ell$, calculate in polynomial time (by Lemma 5.6) the unique position $j = \text{match}(i)$ in $\text{eval}(G)$, and check whether $\text{eval}(G)[i]\text{eval}(G)[j] \notin \text{dom}(R)$. If this is true, then accept; otherwise continue.

3. Calculate in polynomial time (by Lemma 5.8) an SLP G'' that generates $\mathcal{F}(\text{eval}(G)[i+1, j-1])$ and test in polynomial time (by Theorem 4.5) whether $\text{eval}(G'') \xrightarrow{*}_S \varepsilon$. If this is true, then accept; otherwise reject.

This concludes the proof of Theorem 5.4. \square

From Theorems 4.5, 5.2, and 5.4 we obtain the following corollary.

COROLLARY 5.9. *Let (Γ, R) be a 2-homogeneous and confluent presentation.*

Consider the following computational problem:

INPUT: A word $s \in \Gamma^$.*

QUESTION: Does $w \xrightarrow{}_R \varepsilon$ hold?*

If (Γ, R) is N -free, then this problem can be solved in polynomial time; otherwise this problem is coNP-complete.

6. Compressed word problems in PSPACE. In the previous two sections we have investigated 2-homogeneous systems, where every rule is of the form $ab \rightarrow \varepsilon$. In this section we consider a slightly more general class of presentations, where we also allow rules of the form $ab \rightarrow c$. We show that this generalization leads to PSPACE-complete compressed word problems. As a corollary we obtain a fixed deterministic context-free language with a PSPACE-complete compressed word problem, which solves an open problem from [23, 55].

Our PSPACE upper bounds rely all on the following simple fact.

PROPOSITION 6.1. *If the membership problem for the language L (the word problem for a finitely presented monoid \mathcal{M}) belongs to $\bigcup_{c>0} \text{NSPACE}(\log^c(n))$, then the compressed membership problem for L (the compressed word problem for \mathcal{M}) belongs to PSPACE.*

Proof. Assume that the language L belongs to $\text{NSPACE}(\log^c(n))$. Let us fix an SLP G . We decide $\text{eval}(G) \in L$ by simulating the $\text{NSPACE}(\log^c(n))$ algorithm for the membership problem for L on words of length $|\text{eval}(G)|$. Note that a number less than $|\text{eval}(G)|$ can be stored in polynomial space and that for a given position $i \in \{1, \dots, |\text{eval}(G)|\}$ we can calculate $\text{eval}(G)[i]$ in polynomial time. Thus, the simulation gives us a PSPACE-algorithm for the compressed membership problem for L . \square

A presentation (Γ, R) is *weight-reducing* if there exists a weight-function $f : \Gamma^* \rightarrow \mathbb{N}$ such that $f(s) > f(t)$ for all $(s, t) \in R$. Typical examples of weight-reducing presentations are *length-reducing presentations* (i.e., $|s| > |t|$ for all $(s, t) \in R$).

PROPOSITION 6.2. *For every weight-reducing and confluent presentation (Γ, R) , the compressed word problem for $\mathcal{M}(\Gamma, R)$ is in PSPACE.*

Proof. In [41] we have shown that for every fixed weight-reducing and confluent presentation (Γ, R) , the (uncompressed) word problem for $\mathcal{M}(\Gamma, R)$ is in LOGCFL, which is the logspace closure of the class of context-free languages [65]. The class LOGCFL is known to be contained in $\text{NSPACE}(\log^2(n))$ [39]. Thus, membership in PSPACE follows from Proposition 6.1. \square

In the rest of this section, we will show that PSPACE-hardness can be shown already for a quite small subclass of weight-reducing and confluent presentations.

A presentation (Γ, R) is called *monadic* if, for every $(\ell, r) \in R$, $|\ell| > |r|$ and $|r| \leq 1$. A *2-monadic* presentation is a monadic presentation (Γ, R) such that, moreover, $|\ell| = 2$ for every $\ell \in \text{dom}(R)$. In the following, we present a construction that reduces the reachability problem for directed forests to the (uncompressed) word problem of a fixed 2-monadic and confluent presentation (Γ, R) . Later in this section, we will use this construction in order to prove that the compressed word problem for $\mathcal{M}(\Gamma, R)$ is PSPACE-complete.

Let $\Gamma = \{b_0, b_1, c_0, c_1, c_2, \#, \$, \triangleright, 0\}$ and let R be the 2-monadic semi-Thue system consisting of the following rules:

(1) $b_0x \rightarrow \varepsilon$ for all $x \in \{\$, c_0, c_1, c_2\}$	(2) $b_1c_0 \rightarrow \varepsilon$
(3) $b_1\$ \rightarrow \triangleright$	(4) $\triangleright c_i \rightarrow \triangleright$ for all $i \in \{0, 1, 2\}$
(5) $\triangleright\$ \rightarrow \$$	(6) $\#\$ \rightarrow \varepsilon$
(7) $b_1c_2 \rightarrow 0$	
(8) $0x \rightarrow 0$ for all $x \in \Gamma$	(9) $x0 \rightarrow 0$ for all $x \in \Gamma$

Only the rules involving the absorbing symbol 0 produce overlappings. In the resulting critical pairs, both words can be reduced to 0. Thus, R is confluent.

A *directed forest* is a directed acyclic graph (V, E) (where V is the finite set of nodes and $E \subseteq V \times V$ is the edge relation) such that, moreover, for every $u \in V$, $|\{v \in V \mid (u, v) \in E\}| \leq 1$; i.e., every node has at most one outgoing edge. Assume now that (V, E) is a directed forest, where $V = \{v_1, \dots, v_n\}$ and $(v_i, v_j) \in E$ implies $i < j$. Let $v_\alpha \in V$ be a distinguished start node ($1 \leq \alpha \leq n$) and $U \subseteq V$ be a set of final nodes such that every node in U has outdegree 0 (there may be also nodes in $V \setminus U$ without outgoing edges). For $i \leq j$ we define the interval $I_{i,j} = \{v_k \mid i \leq k \leq j\}$. Thus, $I_{1,n} = V$. If $i > j$, we set $I_{i,j} = \emptyset$. We will construct a word $w(v_\alpha, U) \in \Gamma^*$ such that $(v_\alpha, v_i) \in E^*$ for some $v_i \in U$ (i.e., there is a path from the start node to some final node) if and only if $w(v_\alpha, U) \xrightarrow{*}_R 0$, i.e., $w(v_\alpha, U) \xrightarrow{*}_R 0$. For every $i \in \{1, \dots, n\}$ define the word δ_i as follows:

$$\delta_i = \begin{cases} c_0^{n-j+i+1} & \text{if } (v_i, v_j) \text{ is the unique outgoing edge at node } v_i, \\ c_1 & \text{if } v_i \in V \setminus U \text{ and } v_i \text{ has no outgoing edge,} \\ c_2 & \text{if } v_i \in U \text{ (and thus has no outgoing edge).} \end{cases}$$

Note that $\triangleright \delta_i \xrightarrow{*}_R \triangleright$ for all $1 \leq i \leq n$ using the rules in (4). For an interval $I_{i,j}$ ($i \leq j$) we define $\sigma[I_{i,j}] = \delta_i \$ \delta_{i+1} \$ \dots \delta_j \$$. We set $\sigma[\emptyset] = \varepsilon$. Note that $\triangleright \sigma[I_{i,j}] \xrightarrow{*}_R \$ \sigma[I_{i+1,j}]$ if $i \leq j$ using the rules in (4) and (5). Let $\beta = |\sigma[I_{1,\alpha-1}]|$. Finally, define

$$w(v_\alpha, U) = (\#b_1^n)^\beta b_0^\beta \sigma[I_{1,n}].$$

LEMMA 6.3. *We have $w(v_\alpha, U) \xrightarrow{*}_R 0$ if and only if $(v_\alpha, v_i) \in E^*$ for some $v_i \in U$.*

Before we prove Lemma 6.3 let us first consider an example.

Example 6.4. Let (V, E) be the following directed forest (see Figure 6.1). The set U contains only the node v_6 . Let $\alpha = 2$; i.e., v_2 is the start node.

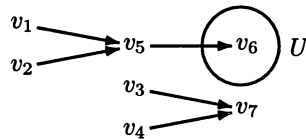


FIG. 6.1.

Then $w(v_\alpha, U) = (\#b_1^7)^7 b_0^5 c_0^4 c_0^5 c_0^4 c_0^5 c_0^7 c_2 c_1$. We obtain the following derivation:

$$\begin{aligned}
& (\#b_1^7)^7 b_0^5 c_0^4 c_0^5 c_0^4 c_0^5 c_0^7 c_2 c_1 \xrightarrow{*}_R && \text{(rules in (1))} \\
& (\#b_1^7)^7 c_0^5 c_0^4 c_0^5 c_0^7 c_2 c_1 \xrightarrow{*}_R && \text{(rule (2))} \\
& (\#b_1^7)^6 \#b_1^2 c_0^4 c_0^5 c_0^7 c_2 c_1 \rightarrow_R && \text{(rule (3))} \\
& (\#b_1^7)^6 \#b_1 \triangleright c_0^4 c_0^5 c_0^7 c_2 c_1 \xrightarrow{*}_R && \text{(rules in (4))} \\
& (\#b_1^7)^6 \#b_1 \triangleright c_0^5 c_0^7 c_2 c_1 \rightarrow_R && \text{(rule (5))} \\
& (\#b_1^7)^6 \#b_1 c_0^5 c_0^7 c_2 c_1 \rightarrow_R && \text{(rule (3))} \\
& (\#b_1^7)^6 \# \triangleright c_0^5 c_0^7 c_2 c_1 \xrightarrow{*}_R && \text{(rules in (4))} \\
& (\#b_1^7)^6 \# \triangleright c_0^7 c_2 c_1 \rightarrow_R && \text{(rule (5))} \\
& (\#b_1^7)^6 \# c_0^7 c_2 c_1 \xrightarrow{*}_R && \text{(rule (6))} \\
& (\#b_1^7)^6 c_0^7 c_2 c_1 \xrightarrow{*}_R && \text{(rule (2))} \\
& (\#b_1^7)^5 \# c_2 c_1 \rightarrow_R && \text{(rule (6))} \\
& (\#b_1^7)^5 c_2 c_1 \xrightarrow{*}_R && \text{(rule (7))} \\
& (\#b_1^7)^4 \#b_1^6 c_1 \xrightarrow{*}_R && \text{(rules in (8) and (9))} \\
& 0.
\end{aligned}$$

Indeed, there exists a path from v_2 to a node in U . If U would consist only of the node v_7 instead of v_6 , then $w(v_\alpha, U) = (\#b_1^7)^7 b_0^5 c_0^4 c_0^5 c_0^4 c_0^5 c_0^7 c_1 c_2$. In this case we obtain a similar derivation showing $w(v_\alpha, U) \xrightarrow{*}_R (\#b_1^7)^5 c_1 c_2$. But the latter word is irreducible. Since R is confluent, $w(v_\alpha, U) \xrightarrow{*}_R 0$ cannot hold.

Proof of Lemma 6.3. First, note that using the rules in (1) we obtain

$$w(v_\alpha, U) = (\#b_1^n)^n b_0^\beta \sigma[I_{1, \alpha-1}] \sigma[I_{\alpha, n}] \xrightarrow{*}_R (\#b_1^n)^n \sigma[I_{\alpha, n}].$$

Claim. For every $v_i \in V$, if $(v_\alpha, v_i) \in E^*$, then there exists $k \geq n - i + \alpha$ such that $w(v_\alpha, U) \xrightarrow{*}_R (\#b_1^n)^k \sigma[I_{i, n}]$.

We prove this claim by induction over the length of the unique path from v_α to v_i . The case $i = \alpha$ is clear. Thus, assume that $(v_\alpha, v_j) \in E^*$ and $(v_j, v_i) \in E$. Then $j < i$ and by induction we have

$$w(v_\alpha, U) \xrightarrow{*}_R (\#b_1^n)^k \sigma[I_{j, n}] = (\#b_1^n)^{k-1} \#b_1^n c_0^{n-i+j+1} \sigma[I_{j+1, n}],$$

where $k \geq n - j + \alpha$, i.e., $k - 1 \geq n - i + \alpha$. We obtain

$$\begin{aligned}
& (\#b_1^n)^{k-1} \#b_1^n c_0^{n-i+j+1} \sigma[I_{j+1, n}] \xrightarrow{*}_R && \text{(rule (2))} \\
& (\#b_1^n)^{k-1} \#b_1^{i-j-1} \sigma[I_{j+1, n}] \rightarrow_R && \text{(rule (3))} \\
& (\#b_1^n)^{k-1} \#b_1^{i-j-2} \triangleright \sigma[I_{j+1, n}] \xrightarrow{*}_R && \text{(rules in (4) and (5))} \\
& (\#b_1^n)^{k-1} \#b_1^{i-j-2} \sigma[I_{j+2, n}] \xrightarrow{*}_R \\
& \quad \vdots \\
& (\#b_1^n)^{k-1} \#b_1^0 \sigma[I_{i, n}] \\
& = (\#b_1^n)^{k-1} \# \sigma[I_{i, n}] \rightarrow_R && \text{(rule (6))} \\
& (\#b_1^n)^{k-1} \sigma[I_{i, n}].
\end{aligned}$$

This proves the claim.

Thus, if $(v_\alpha, v_i) \in E^*$ for some $v_i \in U$, then by the above claim

$$\begin{aligned} w(v_\alpha, U) &\xrightarrow*_R (\#b_1^n)^k \sigma[I_{i,n}] \\ &= (\#b_1^n)^{k-1} \#b_1^{n-1} b_1 c_2 \$\sigma[I_{i+1,n}] \rightarrow_R \quad (\text{rule (7)}) \\ &(\#b_1^n)^{k-1} \#b_1^{n-1} 0 \$\sigma[I_{i+1,n}] \xrightarrow*_R 0 \quad (\text{rules in (8) and (9)}) \end{aligned}$$

for some $k > 0$. On the other hand, if there does not exist $v_i \in U$ with $(v_\alpha, v_i) \in E^*$, then there exists $v_i \in V \setminus U$ with outdegree 0 and $(v_\alpha, v_i) \in E^*$. Thus,

$$w(v_\alpha, U) \xrightarrow*_R (\#b_1^n)^k \sigma[I_{i,n}] = (\#b_1^n)^k c_1 \$\sigma[I_{i+1,n}] \in \text{IRR}(R).$$

Since (Γ, R) is confluent, $w(v_\alpha, U) \xrightarrow*_R 0$ cannot hold. This proves Lemma 6.3. \square

Lemma 6.3 yields the following result that is of independent interest; see section 3.2 for the definition of NC^1 -reductions.

THEOREM 6.5. *There exists a fixed 2-monadic and confluent presentation (Γ, R) such that the word problem for $\mathcal{M}(\Gamma, R)$ is L-hard under NC^1 -reductions.*

Proof. By [18], the reachability problem for directed forests that are ordered (i.e., the set of nodes is $\{1, \dots, n\}$ for some n and $i < j$ whenever there is an edge from i to j) is L-complete under NC^1 -reductions. Moreover, one can assume that 1 is the initial node. It remains to show that for such a forest $G = (V, E)$ and $U \subseteq V$ the word $w(1, U)$ can be constructed in NC^1 from G and U . We leave the details to the reader. \square

The existence of a fixed monadic and confluent presentation with an L-hard word problem was also shown in [6].

Now let us consider the compressed word problem for 2-monadic and confluent presentations.

THEOREM 6.6. *For every 2-monadic and confluent presentation (Γ, R) , the compressed word problem for $\mathcal{M}(\Gamma, R)$ is in PSPACE. There exists a fixed 2-monadic and confluent presentation (Γ, R) such that the compressed word problem for $\mathcal{M}(\Gamma, R)$ is PSPACE-complete.*

Proof. The upper bound follows from Proposition 6.2. For the lower bound we will show that the compressed word problem for the 2-monadic presentation (Γ, R) from the previous discussion is PSPACE-hard. For this we repeat a construction from [41]. Let $\mathcal{A} = (Q, \Sigma, \delta, q_0, q_f)$ be a fixed deterministic linear bounded automaton (Q is the set of states, Σ is the tape alphabet, q_0 (resp., q_f) is the initial (resp., final) state, and $\delta : Q \setminus \{q_f\} \times \Sigma \rightarrow Q \times \Sigma \times \{\text{left}, \text{right}\}$ is the transition function) such that the question of whether a word $w \in \Sigma^*$ is accepted by \mathcal{A} is PSPACE-complete. Such a linear bounded automaton exists; see, e.g., [5]. The one-step transition relation between configurations of \mathcal{A} is denoted by $\Rightarrow_{\mathcal{A}}$. Let $w \in \Sigma^*$ be an input for \mathcal{A} with $|w| = N$. We may assume that \mathcal{A} operates in phases, where a single phase consists of a sequence of $2 \cdot N$ transitions of the form $q_1 \gamma_1 \xrightarrow*_A \gamma_2 q_2 \xrightarrow*_A q_3 \gamma_3$, where $\gamma_1, \gamma_2, \gamma_3 \in \Sigma^N$ and $q_1, q_2, q_3 \in Q$. During the transition sequence $q_1 \gamma_1 \xrightarrow*_A \gamma_2 q_2$, only right-moves are made, whereas during the sequence $\gamma_2 q_2 \xrightarrow*_A q_3 \gamma_3$, only left-moves are made. The automaton \mathcal{A} accepts if it reaches the final state q_f . Otherwise \mathcal{A} does not terminate. There exists a constant $c > 0$ such that if w is accepted by \mathcal{A} , then \mathcal{A} , started on w , reaches the final state q_f after at most $2^{c \cdot N}$ phases. Let $\widehat{\Sigma} = \{\widehat{a} \mid a \in \Sigma\}$ be a disjoint copy of Σ and similarly for \widehat{Q} . Let $\Delta = \Sigma \cup \widehat{\Sigma} \cup \{\langle, 0, 1, \ell\}$ and $\Theta = Q \cup \widehat{Q} \cup \Delta$. We simulate \mathcal{A} by the following semi-Thue system S over the alphabet Θ :

$0\widehat{q} \rightarrow \widehat{q}\mathcal{L}$ for all $q \in Q \setminus \{q_f\}$	$qa \rightarrow \widehat{b}p$ if $\delta(q, a) = (p, b, \text{right})$
$1\widehat{q} \rightarrow 0q$ for all $q \in Q \setminus \{q_f\}$	$\widehat{a}\widehat{q} \rightarrow \widehat{p}b$ if $\delta(q, a) = (p, b, \text{left})$
$q\mathcal{L} \rightarrow 1q$ for all $q \in Q \setminus \{q_f\}$	$q\triangleleft \rightarrow \widehat{q}\triangleleft$ for all $q \in Q \setminus \{q_f\}$

(Θ, S) is length-preserving and if \succ is any linear order on the alphabet Θ that satisfies $Q \succ 1 \succ 0 \succ \widehat{\Sigma} \succ \widehat{Q}$, then $s \succ t$ for every rule $(s, t) \in S$; i.e., S is lexicographic (recall from section 3.1 that we identify an order \succ on the alphabet Θ with its lexicographic extension to Θ^*). Let us choose such a linear order on Θ that, moreover, satisfies $Q \succ \Delta \succ \widehat{Q}$. In [41] we have argued that w is accepted by \mathcal{A} if and only if $1q_0\mathcal{L}^{c \cdot N - 1}w\triangleleft \xrightarrow{*}_S v$ for some word v with $\text{alph}(v) \cap \{q_f, \widehat{q}_f\} \neq \emptyset$. We briefly repeat the arguments: First, note that $1q_0\mathcal{L}^{c \cdot N - 1}w\triangleleft \xrightarrow{*}_S 1^{c \cdot N}q_0w\triangleleft$. From the word $1^{c \cdot N}q_0w\triangleleft$ we can simulate $2^{c \cdot N}$ phases of \mathcal{A} . The crucial point is that the prefix from $\{0, 1\}^*$ acts as a binary counter: for every $u \in \{0, 1\}^i$ ($i < c \cdot N$) and every $q \in Q$ we have $u10^{c \cdot N - |u| - 1}\widehat{q} \xrightarrow{*}_S u1\widehat{q}\mathcal{L}^{c \cdot N - |u| - 1} \rightarrow_S u0q\mathcal{L}^{c \cdot N - |u| - 1} \xrightarrow{*}_S u01^{c \cdot N - |u| - 1}q$. Thus, if \mathcal{A} accepts w , then we can derive from $1q_0\mathcal{L}^{c \cdot N - 1}w\triangleleft$ a word v with $\text{alph}(v) \cap \{q_f, \widehat{q}_f\} \neq \emptyset$. On the other hand, if \mathcal{A} does not accept w and hence does not terminate, then we can derive from $1q_0\mathcal{L}^{c \cdot N - 1}w\triangleleft$ a word of the form $\widehat{q}\mathcal{L}^{c \cdot N}u\triangleleft$ for some $u \in \Sigma^N$ and $q \neq q_f$. Since S is confluent (the left-hand sides of S do not overlap) and $\widehat{q}\mathcal{L}^{c \cdot N}u\triangleleft \in \text{IRR}(S)$, we cannot reach a word v with $\text{alph}(v) \cap \{q_f, \widehat{q}_f\} \neq \emptyset$ from $1q_0\mathcal{L}^{c \cdot N - 1}w\triangleleft$.

To simplify the following construction, we will next expand all rules from S in the following sense: The rule $q\mathcal{L} \rightarrow 1q$, for instance, is replaced by all rules of the form $xq\mathcal{L} \rightarrow x1q$ for all $x \in \Delta$, whereas the rule $0\widehat{q} \rightarrow \widehat{q}\mathcal{L}$ is replaced by all rules of the form $0\widehat{q}x \rightarrow \widehat{q}\mathcal{L}x$ for all $x \in \Delta$. Let us call the resulting system again S . Then S is still length-preserving and lexicographic and $\text{dom}(S) \subseteq \Delta(Q \cup \widehat{Q})\Delta$. The new system S is no longer confluent, but this is not important for the further arguments. It is only important that

$$(6.1) \quad \forall v, v_1, v_2 \in \Delta^*(Q \cup \widehat{Q})\Delta^* : (v_1 \xleftarrow{S} v \xrightarrow{S} v_2) \Rightarrow v_1 = v_2.$$

This is easy to see by inspection of the rules. Moreover, w is accepted by \mathcal{A} if and only if $1q_0\mathcal{L}^{c \cdot N - 1}w\triangleleft \xrightarrow{*}_S v$ for some word v with $\text{alph}(v) \cap \{q_f, \widehat{q}_f\}$. Let $m = (c + 1)N - 1$; thus $m + 3$ is the length of words in any derivation starting from $1q_0\mathcal{L}^{c \cdot N - 1}w\triangleleft$.

Let us now define the directed graph (V, E) , where $V = \bigcup_{i=1}^{m+1} \Delta^i(Q \cup \widehat{Q})\Delta^{m-i+2}$ and $E = \{(v, v') \in V \times V \mid v \xrightarrow{S} v'\}$. This graph is basically the transition graph of the automaton \mathcal{A} on configurations of length N . Since S is lexicographic, (V, E) is acyclic. Moreover, by (6.1), every node from V has at most one outgoing edge. Thus, (V, E) is a directed forest. If we order V lexicographically by \succ and write $V = \{v_1, \dots, v_n\}$ with $v_1 \succ v_2 \succ \dots \succ v_n$, then $(v_i, v_j) \in E$ implies $i < j$. Note that $n = |V| = 2(m + 1) \cdot |Q| \cdot |\Delta|^{m+2}$, which belongs to $2^{O(N)}$. Let U be those words in V that contain either q_f or \widehat{q}_f ; these words have outdegree 0 in the directed forest (V, E) . Let $v_\alpha = 1q_0\mathcal{L}^{c \cdot N - 1}w\triangleleft$. Thus, $\alpha - 1$ is the number of words from V that are lexicographically larger than $1q_0\mathcal{L}^{c \cdot N - 1}w\triangleleft$. The number α can be easily calculated in logarithmic space from the input word w using simple arithmetic. We now have available all the data in order to construct the word $w(v_\alpha, U) \in \Gamma^*$ from Lemma 6.3.

The automaton \mathcal{A} accepts w if and only if there is a path in (V, E) from v_α to a node in U . By Lemma 6.3 this holds if and only if $w(v_\alpha, U) \xleftarrow{*}_R 0$. Thus, it remains to show that the word $w(v_\alpha, U)$ can be generated by a small SLP. Recall the definition

of the words σ_i and $\sigma[I] \in \Gamma^*$, where $1 \leq i \leq n = |V|$ and I is an interval of (V, \succ) that we introduced before Lemma 6.3.

Note that for all $1 \leq i, j \leq n$, if $v_i = u_1 \ell u_2 \rightarrow_S u_1 r u_2 = v_j$ with $(\ell, r) \in S$, then $j - i$ (i.e., the number of words from V that are lexicographically between v_i and v_j) is a number that depends only on the rule (ℓ, r) (and thus ℓ) and $|u_2|$. We call this number $\lambda(\ell, |u_2|)$; it belongs to $2^{O(N)}$ and can be calculated in logarithmic space from ℓ and $|u_2|$ using simple arithmetic. We now describe a small SLP that generates the word $\sigma[V] \in \Gamma^*$. For this let us assume that $Q = \{p_1, \dots, p_{n_1}\}$ and $\Delta = \{a_1, \dots, a_{n_2}\}$ with $p_i \succ p_{i+1}$, $\widehat{p}_i \succ \widehat{p}_{i+1}$, and $a_i \succ a_{i+1}$ (note that the order \succ on the subalphabets Q, \widehat{Q} , and Δ , respectively, is arbitrary except that $1 \succ 0$). We introduce the following productions:⁴

$$\begin{aligned}
 A_i &\rightarrow \prod_{j=1}^{n_2} B_{i,j} A_{i+1} \widehat{B}_{i,j} \text{ for } 0 \leq i < m, \\
 A_m &\rightarrow \prod_{j=1}^{n_2} B_{m,j} \widehat{B}_{m,j}, \\
 B_{i,j} &\rightarrow \prod_{k=1}^{n_1} \prod_{\ell=1}^{n_2} (C_{i,j,k,\ell} \mathbb{S})^{|\Delta|^{m-i}} \text{ for } 0 \leq i \leq m, 1 \leq j \leq n_2, \\
 C_{i,j,k,\ell} &\rightarrow \begin{cases} c_0^{n-\lambda(a_j p_k a_\ell, m-i)+1} & \text{if } a_j p_k a_\ell \in \text{dom}(R), \\ c_1 & \text{if } a_j p_k a_\ell \notin \text{dom}(R) \text{ and } p_k \neq q_f, \\ c_2 & \text{if } p_k = q_f, \end{cases} \\
 \widehat{B}_{i,j} &\rightarrow \prod_{k=1}^{n_1} \prod_{\ell=1}^{n_2} (\widehat{C}_{i,j,k,\ell} \mathbb{S})^{|\Delta|^{m-i}} \text{ for } 0 \leq i \leq m, 1 \leq j \leq n_2, \\
 \widehat{C}_{i,j,k,\ell} &\rightarrow \begin{cases} c_0^{n-\lambda(a_j \widehat{p}_k a_\ell, m-i)+1} & \text{if } a_j \widehat{p}_k a_\ell \in \text{dom}(R), \\ c_1 & \text{if } a_j \widehat{p}_k a_\ell \notin \text{dom}(R) \text{ and } \widehat{p}_k \neq \widehat{q}_f, \\ c_2 & \text{if } \widehat{p}_k = \widehat{q}_f. \end{cases}
 \end{aligned}$$

The exponents that appear in the right-hand sides of the productions for the nonterminal $B_{i,j}$ and $\widehat{B}_{i,j}$, namely $|\Delta|^{m-i}$, are of size $2^{O(N)}$ and can therefore be replaced by sequences of ordinary productions. Note that $\text{eval}(C_{i,j,k,\ell}) = \delta_s$ for every node v_s from $\Delta^i a_j p_k a_\ell \Delta^{m-i}$, whereas $\text{eval}(\widehat{C}_{i,j,k,\ell}) = \delta_s$ for every node v_s from $\Delta^i a_j \widehat{p}_k a_\ell \Delta^{m-i}$. It follows that for all $0 \leq i \leq m$, all $u \in \Delta^i$, and all $1 \leq j \leq n_2$ we have (note that $ua_j Q \Delta^{m-i+1} \subseteq V$ is an interval of (V, \succ))

$$(6.2) \quad \text{eval}(B_{i,j}) = \sigma[ua_j Q \Delta^{m-i+1}] \quad \text{and} \quad \text{eval}(\widehat{B}_{i,j}) = \sigma[ua_j \widehat{Q} \Delta^{m-i+1}].$$

Claim. Let $0 \leq i \leq m$ and let $u \in \Delta^i$ be arbitrary. Then for the interval $I = \bigcup_{j=1}^{m-i+1} u \Delta^j (Q \cup \widehat{Q}) \Delta^{m-i-j+2}$ of the linear order (V, \succ) we have $\sigma[I] = \text{eval}(A_i)$.

The claim will be shown by induction on i (for $i = m$ down to 0). For $i = m$ the

⁴The expression $\prod_{i=1}^k w_i$ is an abbreviation for $w_1 w_2 \dots w_k$.

claim is true:

$$\begin{aligned} \text{eval}(A_m) &= \prod_{j=1}^{n_2} \text{eval}(B_{m,j})\text{eval}(\widehat{B}_{m,j}) \quad (\text{by (6.2)}) \\ &= \prod_{j=1}^{n_2} \sigma[ua_jQ\Delta]\sigma[ua_j\widehat{Q}\Delta] \quad (Q \succ \widehat{Q}) \\ &= \sigma[u\Delta(Q \cup \widehat{Q})\Delta]. \end{aligned}$$

Now let $i < m$ and $u \in \Delta^i$. The words from the interval $\bigcup_{j=1}^{m-i+1} u\Delta^j(Q \cup \widehat{Q})\Delta^{m-i-j+2}$ can be partitioned into the following decreasing sequence of consecutive intervals of (V, \succ) (recall that $Q \succ \Delta \succ \widehat{Q}$ and $a_1 \succ a_2 \succ \dots \succ a_{n_2}$):

$$\begin{aligned} ua_1Q\Delta^{m-i+1} &\succ \bigcup_{j=1}^{m-i} ua_1\Delta^j(Q \cup \widehat{Q})\Delta^{m-i-j+1} \succ ua_1\widehat{Q}\Delta^{m-i+1} \\ \succ ua_2Q\Delta^{m-i+1} &\succ \bigcup_{j=1}^{m-i} ua_2\Delta^j(Q \cup \widehat{Q})\Delta^{m-i-j+1} \succ ua_2\widehat{Q}\Delta^{m-i+1} \\ \succ \dots \succ ua_{n_2}Q\Delta^{m-i+1} &\succ \bigcup_{j=1}^{m-i} ua_{n_2}\Delta^j(Q \cup \widehat{Q})\Delta^{m-i-j+1} \succ ua_{n_2}\widehat{Q}\Delta^{m-i+1}. \end{aligned}$$

By induction, we have

$$\begin{aligned} &\sigma \left[\bigcup_{j=1}^{m-i} ua_1\Delta^j(Q \cup \widehat{Q})\Delta^{m-i-j+1} \right] \\ &= \dots = \sigma \left[\bigcup_{j=1}^{m-i} ua_{n_2}\Delta^j(Q \cup \widehat{Q})\Delta^{m-i-j+1} \right] = \text{eval}(A_{i+1}). \end{aligned}$$

Together with (6.2) we obtain

$$\sigma \left[\bigcup_{j=1}^{m-i+1} u\Delta^j(Q \cup \widehat{Q})\Delta^{m-i-j+2} \right] = \prod_{j=1}^{n_2} \text{eval}(B_{i,j})\text{eval}(A_{i+1})\text{eval}(\widehat{B}_{i,j}) = \text{eval}(A_i).$$

This proves the claim. By setting $i = 0$ we get

$$\text{eval}(A_0) = \sigma \left[\bigcup_{j=1}^{m+1} \Delta^j(Q \cup \widehat{Q})\Delta^{m-j+2} \right] = \sigma[V].$$

Let $\beta = |\sigma[I_{1,\alpha-1}]| \in 2^{O(N)}$. Arithmetic on numbers with $N^{O(1)}$ many bits allows us to compute β in logspace from the input word w . Using the above productions and the number β , we can construct an SLP G of size polynomial in the input size N with $\text{eval}(G) = (\#b_1^n)^n b_0^\beta \sigma[V] = w(v_\alpha, U)$. Recall that n is of size $2^{O(N)}$. Then our input word w is accepted by \mathcal{A} if and only if $\text{eval}(G) \stackrel{*}{\leftrightarrow}_R 0$. This proves the theorem. \square

The following corollary solves an open problem from [23, 55].

COROLLARY 6.7. *There exists a fixed deterministic context-free language L such that the compressed membership problem for L is PSPACE-complete.*

Proof. Since every context-free language is contained in $\text{DSPACE}(\log^2(n))$, the PSPACE upper bound can be deduced from Proposition 6.1. For the lower bound,

notice that the language $\{w \in \Gamma^* \mid w \xrightarrow{*}_R 0\}$ is deterministic context-free for every monadic and confluent presentation and every $0 \in \Gamma$; see, e.g., [11, Theorem 4.2.7]. If we choose the 2-monadic and confluent presentation from the proof of Theorem 6.5 for (Γ, R) , then the language $\{w \in \Gamma^* \mid w \xrightarrow{*}_R 0\}$ is PSPACE-hard by the proof of Theorem 6.6. \square

In [31] a language L is called *deterministic linear* if it is accepted by a deterministic 1-turn pushdown automaton.

COROLLARY 6.8. *There exists a fixed deterministic linear language L such that the compressed membership problem for L is PSPACE-complete.*

Proof. The language $\{w \in \Gamma^* \mid w \xrightarrow{*}_R 0\} \cap (\#b_1^+)^+ b_0^+ ((c_0^+ \cup c_1 \cup c_2)\$)^+$ is easily seen to be deterministic linear. Moreover, it contains a word of the form $w(v_\alpha, U)$ if and only if $w(v_\alpha, U) \xrightarrow{*}_R 0$. \square

7. Compressed word problems in EXPSPACE. The largest class of monoid presentations that we consider in this paper are weight-lexicographic and confluent presentations: A presentation (Γ, R) is *weight-lexicographic* if there exist a linear order \succ on the alphabet Γ and a weight-function $f : \Gamma^* \rightarrow \mathbb{N}$ such that for all $(s, t) \in R$ we have either $f(s) > f(t)$ or $(f(s) = f(t) \text{ and } s \succ t)$. If the weight-function f is the length-function, i.e., $f(w) = |w|$, then (Γ, R) is called *length-lexicographic*.

THEOREM 7.1. *For every weight-lexicographic and confluent presentation (Γ, R) , the compressed word problem for $\mathcal{M}(\Gamma, R)$ is in EXPSPACE. Moreover, there exists a fixed length-lexicographic and confluent presentation (Γ, R) such that the compressed word problem for $\mathcal{M}(\Gamma, R)$ is EXPSPACE-complete.*

Proof. For the upper bound we can use a result from [41]: for every fixed weight-lexicographic and confluent presentation (Γ, R) , the word problem for $\mathcal{M}(\Gamma, R)$ belongs to PSPACE. Thus, for two given SLPs G_1 and G_2 we can first generate the exponentially long words $\text{eval}(G_1)$ and $\text{eval}(G_2)$ and then check in space bounded polynomially in $|\text{eval}(G_1)| + |\text{eval}(G_2)|$ whether $\text{eval}(G_1) \xleftrightarrow{*}_R \text{eval}(G_2)$.

For the lower bound, let (Θ, S) be the presentation from the proof of Theorem 6.6, where this time the simulated machine \mathcal{A} is a fixed Turing-machine that accepts an EXPSPACE-complete language. Also for such a machine we may assume that it operates in alternating phases of left and right sweeps. The presentation (Θ, S) is length-lexicographic and confluent. Without loss of generality the space bound for an input of length n is 2^n . The number of phases can be bounded by $2^{c \cdot 2^n}$ for some constant $c > 0$. Add to Θ an absorbing symbol 0 and add to S the following rules: $q_f \rightarrow 0$, $\hat{q}_f \rightarrow 0$, $x0 \rightarrow 0$ for all $x \in \Theta$, and $0x \rightarrow 0$ for all $x \in \Theta$. We again call the resulting presentation (Θ, S) ; it is still length-lexicographic and confluent. Moreover, for an input word w of length n , w is accepted by \mathcal{A} if and only if $1q_0 \mathcal{L}^{c \cdot 2^n - 1} w \square^{2^n - |w|} \triangleleft \xleftrightarrow{*}_S 0$ (where q_0 is the initial state of \mathcal{A} and \square is the blank symbol); see also the proof of Theorem 6 in [41]. Finally, note that the word $1q_0 \mathcal{L}^{c \cdot 2^n - 1} w \square^{2^n - |w|} \triangleleft$ can be generated by an SLP of polynomial size in n . \square

The language $\{w \in \Theta^* \mid w \xrightarrow{*}_S 0\}$, where (Θ, S) is the presentation from the previous proof, is context-sensitive. Thus, we obtain the following result.

COROLLARY 7.2. *There exists a fixed context-sensitive language L such that the compressed membership problem for L is EXPSPACE-complete.*

8. Circuit complexity and compression. In this section we will investigate the compressed membership problem for languages from very low complexity classes. These classes are usually defined by uniform families of small depth Boolean circuit families. An equivalent and for our purpose more suitable definition is based on

alternating Turing-machines with logarithmic time bounds; see section 3.2. Recall that ALOGTIME denotes the class of all languages that can be recognized on an alternating Turing-machine in time $O(\log(n))$. Within ALOGTIME, we can define the logtime hierarchy: For $k \geq 1$ we denote by Σ_k^{\log} (resp., Π_k^{\log}) the class of all languages that can be decided by an alternating Turing-machine in time $O(\log(n))$ within $k - 1$ alternations (on every computation path) starting in an existential state (resp., universal state). In [4], $\Sigma_k^{\log} \cup \Pi_k^{\log}$ is proposed as a uniform version of AC_k^0 , which is the class of all languages that can be recognized by a polynomial-size, depth k family of unbounded fan-in Boolean circuits. The union $\bigcup_{k \geq 0} \Sigma_k^{\log} \cup \Pi_k^{\log}$ is also called the *logtime hierarchy* (LH). By [63], LH is a strict hierarchy.

The well-known *polynomial time hierarchy* [64] is defined similarly to the LH: For $k \geq 1$ we denote by Σ_k^p (resp., Π_k^p) the class of all languages that can be decided by an alternating Turing-machine in polynomial time within $k - 1$ alternations (on every computation path) starting in an existential state (resp., universal state).

For the further investigations we will need the following lemma.

LEMMA 8.1. *Incrementing a binary n -bit counter C (with arbitrary initial value) m times by 1 takes a total of $O(n + m)$ steps on a deterministic Turing-machine.*

Proof. Without loss of generality assume that m is a power of 2. We assume that after each increment, the read-write head moves back to the least significant bit of C . If the first i bits of C are $1^{i-1}0$ in that order, then we need $2i$ steps for the increment. For $i > \log(m)$, this will happen only once during the m increments. Thus, it suffices to show that incrementing a binary $\log(m)$ -bit counter m times by 1 takes a total of $O(m)$ steps. There are at most $2^{\log(m)-i} = \frac{m}{2^i}$ numbers $t \in \{0, \dots, m\}$ such that the first i bits of t are $1^{i-1}0$ in that order. Incrementing such a number by 1 takes $2i$ steps. Thus, incrementing the counter m times by 1 takes a total of $\sum_{i=1}^{\log(m)} 2i \cdot \frac{m}{2^i} \leq 2m \sum_{i=1}^{\infty} \frac{i}{2^i} = 4m$ steps. \square

Of course, an analogous statement for decrementing a counter is also true.

THEOREM 8.2. *For every language L in Σ_k^{\log} (resp., Π_k^{\log}) the compressed membership problem belongs to Σ_k^p (resp., Π_k^p). Moreover, there exists a fixed language L in Σ_k^{\log} (resp., Π_k^{\log}) such that the compressed membership problem for L is Σ_k^p -complete (resp., Π_k^p -complete).*

Proof. It suffices to prove the statement for Σ_k^p and Σ_k^{\log} , respectively, because Π_k^p and Π_k^{\log} are the corresponding complementary classes. Let us first show that the compressed membership problem for L belongs to Σ_k^p in case L belongs to Σ_k^{\log} . Assume that L belongs to Σ_k^{\log} . Given an SLP G we simulate the Σ_k^{\log} -algorithm on $\text{eval}(G)$. Since $|\text{eval}(G)| \in 2^{O(n)}$, this simulation takes $O(n)$ steps. Moreover, the number of alternations during the simulation is k and we start in an existential state. Finally, note that if the Σ_k^{\log} -machine has written a position $i \in \{1, \dots, |\text{eval}(G)|\}$ on its address tape and queries the i th position of $\text{eval}(G)$, then in the simulation we have to determine the symbol $\text{eval}(G)[i]$ which is possible in polynomial time (with respect to $|G|$).

Next, we will construct a language L in Σ_k^{\log} such that the compressed membership problem for L is Σ_k^p -complete. First, assume that k is odd. In this case the following restricted version of quantified Boolean satisfiability (QBF), called Q3SAT $_k$, is Σ_k^p -complete [70]:

INPUT: A quantified Boolean formula of the form

$$\Theta = \exists \bar{x}_1 \forall \bar{x}_2 \cdots \exists \bar{x}_k : \varphi(x_1, \dots, x_n).$$

Here $\bar{x}_i = (x_{\ell_i}, \dots, x_{\ell_{i+1}-1})$ (where $\ell_1 = 1$ and $\ell_{k+1} = n + 1$) is a sequence of Boolean variables and φ is a formula in 3-CNF (a conjunction of disjunctions, each containing exactly three literals) over the variables x_1, \dots, x_n .

QUESTION: Is Θ a true quantified Boolean formula?

Let us take an instance Θ of Q3SAT $_k$ of the above form. Assume that $\varphi = C_1 \wedge C_2 \wedge \dots \wedge C_m$ where every C_i is a disjunction of three literals. Note that there are 2^n truth assignments to the variables x_1, \dots, x_n and each of these truth assignments can be represented by the vector (b_1, \dots, b_n) where $b_i = 1$ if the variable x_i evaluates to true; otherwise $b_i = 0$. We order these vectors lexicographically, where the last position gets the highest significance, i.e., $(0, \dots, 0) < (1, 0, \dots, 0) < \dots < (1, \dots, 1)$. Thus, we can speak of the j th truth assignment ($0 \leq j \leq 2^n - 1$). For each disjunction C_i define the word $c_i = b_0 b_1 \dots b_{2^n-1}$, where $b_j \in \{0, 1\}$ is the truth value of C_i under the j th truth assignment. In [8] it is shown that the word c_i can be generated by an SLP of size $O(n)$. Let $d_i = \ell_{i+1} - \ell_i$; i.e., d_i is the number of variables in the i th block \bar{x}_i . Let $\Gamma = \{0, 1, \$, a_1, \dots, a_k\}$. Finally, we define the word $w(\Theta) \in \Gamma^*$ by

$$w(\Theta) = c_1 c_2 \dots c_m \$^{2^m} a_k^{d_k} \dots a_2^{d_2} a_1^{d_1}.$$

Since every c_i can be generated by an SLP of size $O(n)$, the word $w(\Theta)$ can be generated by an SLP of polynomial size with respect to the size of the formula Θ . Note that $|w(\Theta)| = m \cdot 2^n + 2^m + n$. Thus $n \leq \log(|w(\Theta)|)$ and also $m \leq \log(|w(\Theta)|)$. The only use of the padding-factor $\$^{2^m}$ is to ensure $m \leq \log(|w(\Theta)|)$. It remains to construct a Σ_k^{\log} -machine \mathcal{A} that accepts a given input word of the form $w(\Theta)$ if and only if Θ is true. The behavior of this machine on inputs that are not of the form $w(\Theta)$ is not important; it is important only that the logarithmic time bound is respected, independently of the form of the input. This will be ensured by a counter t . In the following, we write $w[i]$ ($i \in \{0, \dots, |w| - 1\}$), where i is the current content of the address tape, for the result of querying the input w via the random access mechanism. Note that in contrast to the definition in section 3.1, we number the first position of w with 0 and the last position with $|w| - 1$. The alternating Turing-machine \mathcal{A} is described in Figure 8.1. Let us consider an example before we continue analyzing the machine \mathcal{A} .

Example 8.3. Let Θ be the quantified Boolean formula

$$\exists x_1 \forall x_2 \exists x_3 : (x_1 \vee x_2 \vee \neg x_3) \wedge (\neg x_1 \vee x_2 \vee x_3),$$

which evaluates to true. We have $c_1 = 11110111$ and $c_2 = 10111111$ and thus

$$w(\Theta) = 1111011110111111\$^4 a_3 a_2 a_1.$$

Consider the truth assignment $x_1 = x_2 = 1, x_3 = 0$. This is the third assignment in lexicographic order (recall that we start with the 0th assignment). The fact that $(x_1 \vee x_2 \vee \neg x_3) \wedge (\neg x_1 \vee x_2 \vee x_3)$ is true under this assignment corresponds to the fact that $w(\Theta)[3] = 1$ and $w(\Theta)[3 + 2^m] = w(\Theta)[11] = 1$. This is verified by the machine \mathcal{A} in the second while-loop, where $p = 3$ is guessed in the for-loop.

Now let us analyze the behavior of the alternating Turing-machine \mathcal{A} on an input $w \in \Gamma^*$. All counters (t, s, n , and q) in this algorithm need only $\log(|w|)$ bits and are incremented (or decremented) only $O(\log(|w|))$ times. Thus, by Lemma 8.1 all increments and decrements for the various counters need $O(\log(|w|))$ total time. Hence, the counter t enforces a logarithmic time bound of the whole algorithm. Since the number of alternations is precisely k and \mathcal{A} starts in an existential state, the

```

input a string  $w \in \Gamma^*$ 
 $t := 2\lceil \log(|w|) \rceil$  (by [3, Lemma 6.1],  $t$  can be calculated in DLOGTIME)
 $s := |w| - 1$  (again, by [3, Lemma 6.1],  $s$  can be calculated in DLOGTIME)
 $p := 0; n := 0;$ 
for  $i = 1$  to  $k$  do
  while  $w[s] = a_i$  and  $t > 0$  do
     $t := t - 1; s := s - 1; n := n + 1;$ 
    Guess existentially (if  $i$  is odd) or universally (if  $i$  is even) the  $n$ th bit of
    the number  $p$ .
  endwhile
endfor
Copy the  $n$ -bit number  $p$  to the address tape, initialize a counter  $q$  to 0, and
append  $q$  to the  $n$  bits of the address tape. When querying the input tape
via the address tape, the machine  $\mathcal{A}$  will interpret the content of the address
tape (i.e., the concatenation of the  $n$  bits of  $p$  followed by the bits of  $q$ ) as the
number  $p + 2^n \cdot q$ .
while  $w[p + 2^n \cdot q] = 1$  and  $t > 0$  do
   $q := q + 1; t := t - 1;$ 
endwhile
if  $w[p + 2^n \cdot q] = 0$  then reject else accept

```

FIG. 8.1. *The alternating Turing-machine \mathcal{A} .*

machine \mathcal{A} is indeed a Σ_k^{\log} -machine. We claim that if the input w is of the form $w(\Theta)$ for an instance Θ of Q3SAT_k , then the counter t does not reach the value 0 if \mathcal{A} is started on $w(\Theta)$. Let us fix an instance Θ of Q3SAT_k with n variables and m clauses. Thus, $n \leq \log(|w(\Theta)|)$ and $m \leq \log(|w(\Theta)|)$. From the construction of $w(\Theta)$ it follows that \mathcal{A} decrements the counter t only $n + m \leq 2\lceil \log(|w(\Theta)|) \rceil$ times when \mathcal{A} runs on the input $w(\Theta)$. Thus, t does not reach the value 0. Using this observation, it is easy to see that \mathcal{A} accepts $w(\Theta)$ if and only if Θ is true. This finishes the presentation of a language in Σ_k^{\log} with a Σ_k^p -complete compressed membership problem for the case that k is odd. If k is even, one can argue analogously; one has to only replace the 3-CNF formula by a 3-DNF formula. The resulting variant of Q3SAT_k is Σ_k^p -complete [70]. \square

Let us remark that the logtime hierarchy LH can be also characterized using first-order logic (FO) with ordering and the BIT predicate: $\text{LH} = \text{FO}[<, \text{BIT}]$; see, e.g., [32] for definitions. Since, for instance, NP can be captured by existential second-order logic ($\text{NP} = \text{SO}\exists$), it follows from Theorem 8.2 that $\text{FO}[<, \text{BIT}]$ properties on strings cannot be translated into $\text{SO}\exists$ properties on SLPs unless the polynomial time hierarchy collapses. In a setting without the BIT predicate, similar definability issues are investigated in [1].

By Proposition 6.1, for every language in $\bigcup_{i \geq 0} \text{NSPACE}(\log^i(n))$ the compressed membership problem belongs to PSPACE. It turns out that we find languages with a PSPACE-complete compressed membership problem already in $\text{ALOGTIME} \subseteq \text{DSPACE}(\log(n))$.

THEOREM 8.4. *There exists a fixed language L in ALOGTIME such that the compressed membership problem for L is PSPACE-complete.*

Proof. We can reuse the construction from the previous proof, except that we start from an instance of QBF which is PSPACE-complete [52]. \square

One should note that it is *not* the case that for every ALOGTIME-complete language the compressed membership problem is PSPACE-complete (unless $P = PSPACE$): The word problem for the finite group S_5 is ALOGTIME-complete [2], but the compressed word problem for S_5 is in P ; in fact, it is P -complete [7]. Another example is the word problem for the free group F_2 of rank 2. Its compressed membership problem is P -complete by Theorem 4.9, whereas the word problem for F_2 is ALOGTIME-hard [57] but not even known to be in ALOGITME. Thus, in the framework of SLPs, a general upgrading theorem analogous to [68] does not hold. The next theorem states this fact in a more general context. For this, we introduce a parallel notion of reducibility that we call LOGDCFL-reducibility.

A deterministic logspace-bounded AuxPDA is a deterministic pushdown automaton that has an auxiliary read-write working tape of length $O(\log(n))$ for an input of length n [65]. It is known that a language can be recognized by a deterministic logspace-bounded AuxPDA in polynomial time if and only if it belongs to the class LOGDCFL, which is the class of all problems that are logspace reducible to a deterministic context-free language [65]. We say that a function $f : \Gamma^* \rightarrow \Sigma^*$ is computable in LOGDCFL if there exists a deterministic logspace-bounded AuxPDA with an output tape that computes for an input $x \in \Gamma^*$ the word $f(x)$ on the output tape in polynomial time. This leads to the notion of LOGDCFL-reductions. LOGDCFL-reducibility is denoted by \leq_{LOGDCFL} . It is easy to see that a LOGDCFL-computable function belongs to the functional class L^{LOGDCFL} of [27], which is the class of all functions that can be computed by a logspace transducer which has additional access to an oracle from LOGCFL. Since L^{LOGDCFL} is contained in functional NC^2 [27], we see that LOGDCFL-computable functions also belong to functional NC^2 .

If \mathcal{R} is any notion of reducibility, then we write $A \equiv_{\mathcal{R}} B$ for $A \leq_{\mathcal{R}} B \leq_{\mathcal{R}} A$. In the following proposition, we denote for a language K by $\mathcal{C}(K)$ the compressed membership problem for K .

PROPOSITION 8.5. *For every language L there exists a language K such that $L \equiv_{\text{NC}^1} K \equiv_{\text{LOGDCFL}} \mathcal{C}(K)$.*

Proof. Let us fix a language $L \subseteq \Gamma^*$, let $\# \notin \Gamma$ be a new symbol, and define

$$K = \{a_1 \# a_2 \#^2 a_3 \cdots \#^{n-1} a_n \mid a_1 a_2 \cdots a_n \in L, a_1, \dots, a_n \in \Gamma\}.$$

Then $L \equiv_{\text{NC}^1} K$ is easy to see.

That K is LOGDCFL-reducible to $\mathcal{C}(K)$ is trivial. Thus, it remains to show $\mathcal{C}(K) \leq_{\text{LOGDCFL}} K$. Note that if an SLP G generates a word $a_1 \# a_2 \#^2 a_3 \cdots \#^{n-1} a_n$ (which has length $\frac{n(n+1)}{2}$), then $|G| \geq n$. This is true, because if for a nonterminal A , $\text{eval}_G(A)$ contains more than one symbol from Γ , then A can occur only once in the whole derivation tree generated by G . Now a LOGDCFL-reduction from $\mathcal{C}(K)$ to K can be implemented as follows: For a given SLP G , a deterministic logspace-bounded AuxPDA generates the word $\text{eval}(G)$ on the pushdown in the same way as context-free languages are recognized on pushdown automata. Moreover, every time the AuxPDA pushes a terminal on the pushdown, it writes that terminal on the output tape, increments a counter by 1, and removes the terminal from the pushdown. If the counter reaches the value $\frac{|G|(|G|+1)}{2} + 1$ (which has $O(\log(|G|))$ many bits in its binary representation), then the AuxPDA terminates immediately (this ensures a polynomial running time) and writes, for instance, $\#\#$ on the output in order to ensure that the generated output does not belong to K . If the counter does not reach the value $\frac{m(m+1)}{2} + 1$, then the AuxPDA finally has produced the word $\text{eval}(G)$ on the output. We have described a LOGDCFL-reduction from $\mathcal{C}(K)$ to K . \square

From Proposition 8.5 it follows that if C is a complexity class that is closed under NC^1 -reductions and such that C has complete problems under some notion \mathcal{R} of reducibility that is weaker than LOGDCFL-reducibility (e.g., NC^2 -reducibility), then C contains a language L such that both L and $\mathcal{C}(L)$ are complete for C under \mathcal{R} -reducibility.

We remark that also for hierarchical graph descriptions [38] (which can be viewed as graph-generating SLPs) the correlation between the complexity of a problem in its compressed and uncompressed variant, respectively, is quite loose.

9. Uniform variants. Many of the decision problems in this paper can be also investigated in a uniform setting. For a class \mathcal{C} of monoid presentations define the *compressed uniform word problem* for the class \mathcal{C} as the following decision problem:

INPUT: A monoid presentation (Γ, R) and two SLPs G_1 and G_2 over the terminal alphabet Γ .

QUESTION: Does $\text{eval}(G_1) \stackrel{*}{\leftrightarrow}_R \text{eval}(G_2)$ hold?

Similarly we can define the *compressed uniform membership problem* for a class \mathcal{C} of languages. Here we have to specify the representation of a language from \mathcal{C} . For various representations of regular languages, the complexity of the uniform compressed membership problem was investigated in [55]. The upper bound in the following result was also stated in [55].

THEOREM 9.1. *The compressed uniform membership problem for the class of all context-free languages (represented by context-free grammars) is PSPACE-complete.*

Proof. The lower bound follows from Corollary 6.7. For the upper bound it was argued in [55] that one can use a $DSPACE(\log^2(n))$ algorithm for parsing context-free languages in the same way as in the proof of Corollary 6.7. But here, a problem arises: The uniform membership problem for context-free languages is P-complete, and thus probably not contained in $\bigcup_{c>0} DSPACE(\log^c(n))$. On the other hand, by [26] the uniform membership problem for context-free grammars in *Chomsky normal form* can be solved in $DSPACE(\log^2(|G| + |w|))$, where $|G|$ is the size of the input grammar and w is the word that has to be tested for membership. Since every context-free grammar can be transformed in polynomial time into Chomsky normal form, we can argue analogously to the proof of Proposition 6.1. \square

REFERENCES

- [1] F. AFRATI, H. LEISS, AND M. DE ROUGEMONT, *Definability and compression*, Fund. Inform., 56 (2003), pp. 155–180.
- [2] D. A. M. BARRINGTON, *Bounded-width polynomial-size branching programs recognize exactly those languages in NC^1* , J. Comput. System Sci., 38 (1989), pp. 150–164.
- [3] D. A. M. BARRINGTON, N. IMMERMANN, AND H. STRAUBING, *On uniformity within NC^1* , J. Comput. System Sci., 41 (1990), pp. 274–306.
- [4] D. A. M. BARRINGTON, C.-J. LU, P. B. MILTERSEN, AND S. SKYUM, *Searching constant width mazes captures the AC^0 hierarchy*, in Proceedings of the 15th Annual Symposium on Theoretical Aspects of Computer Science (STACS 98), Paris, France, Lecture Notes in Comput. Sci. 1373, M. Morvan, C. Meinel, and D. Krob, eds., Springer, Berlin, 1998, pp. 73–83.
- [5] G. BAUER AND F. OTTO, *Finite complete rewriting systems and the complexity of the word problem*, Acta Inform., 21 (1984), pp. 521–540.
- [6] M. BEAUDRY, M. HOLZER, G. NIEMANN, AND F. OTTO, *McNaughton families of languages*, Theoret. Comput. Sci., 290 (2003), pp. 1581–1628.
- [7] M. BEAUDRY, P. MCKENZIE, P. PÉLADÉAU, AND D. THÉRIEN, *Finite monoids: From word to circuit evaluation*, SIAM J. Comput., 26 (1997), pp. 138–152.

- [8] P. BERMAN, M. KARPINSKI, L. L. LARMORE, W. PLANDOWSKI, AND W. RYTTER, *On the complexity of pattern matching for highly compressed two-dimensional texts*, J. Comput. System Sci., 65 (2002), pp. 332–350.
- [9] R. V. BOOK, *Homogeneous Thue systems and the Church–Rosser property*, Discrete Math., 48 (1984), pp. 137–145.
- [10] R. V. BOOK, M. JANTZEN, B. MONIEN, C. P. Ó’DÚNLAING, AND C. WRATHALL, *On the complexity of word problems in certain Thue systems*, in Proceedings of the 10th International Symposium on Mathematical Foundations of Computer Science (MFCS’81), Štrbské Pleso, Czechoslovakia, Lecture Notes in Comput. Sci. 118, J. Gruska and M. Chytil, eds., Springer, Berlin, New York, 1981, pp. 216–223.
- [11] R. V. BOOK AND F. OTTO, *String–Rewriting Systems*, Springer, New York, 1993.
- [12] W. W. BOONE, *The word problem*, Ann. of Math. (2), 70 (1959), pp. 207–265.
- [13] B. BORCHERT AND A. LOZANO, *Succinct circuit representations and leaf language classes are basically the same concept*, Inform. Process. Lett., 59 (1996), pp. 211–215.
- [14] G. BUSATTO, M. LOHREY, AND S. MANETH, *Efficient memory representation of XML documents*, in Proceedings of the 10th International Symposium on Database Programming Languages (DBPL 2005), Trondheim, Norway, Lecture Notes in Comput. Sci. 3774, G. M. Bierman and Ch. Koch, eds., Springer, Berlin, 2005, pp. 199–216.
- [15] A. K. CHANDRA, D. C. KOZEN, AND L. J. STOCKMEYER, *Alternation*, J. Assoc. Comput. Mach., 28 (1981), pp. 114–133.
- [16] M. CHARIKAR, E. LEHMAN, D. LIU, R. PANIGRAHY, M. PRABHAKARAN, A. RASALA, A. SAHAI, AND A. SHELAT, *Approximating the smallest grammar: Kolmogorov complexity in natural models*, in Proceedings of the 34th Annual Symposium on Theory of Computing (STOC 2002), Montréal, Canada, ACM, New York, 2002, pp. 792–801.
- [17] S. A. COOK, *A taxonomy of problems with fast parallel algorithms*, Inform. and Control, 64 (1985), pp. 2–22.
- [18] S. A. COOK AND P. MCKENZIE, *Problems complete for deterministic logarithmic space*, J. Algorithms, 8 (1987), pp. 385–394.
- [19] M. FARACH AND M. THORUP, *String matching in Lempel–Ziv compressed strings*, Algorithmica, 20 (1998), pp. 388–404.
- [20] J. FEIGENBAUM, S. KANNAN, M. Y. VARDI, AND M. VISWANATHAN, *The complexity of problems on graphs represented as OBDDs*, Chicago J. Theoret. Comput. Sci., 1999, article 5.
- [21] H. GALPERIN AND A. WIGDERSON, *Succinct representations of graphs*, Inform. and Control, 56 (1983), pp. 183–198.
- [22] M. R. GAREY AND D. S. JOHNSON, *Computers and Intractability: A Guide to the Theory of NP–Completeness*, W. H. Freeman, San Francisco, 1979.
- [23] L. GASIENIEC, M. KARPINSKI, W. PLANDOWSKI, AND W. RYTTER, *Efficient algorithms for Lempel–Ziv encoding (extended abstract)*, in Proceedings of the 5th Scandinavian Workshop on Algorithm Theory (SWAT 1996), Reykjavik, Iceland, Lecture Notes in Comput. Sci. 1097, R. G. Karlsson and A. Lingas, eds., Springer, Berlin, 1996, pp. 392–403.
- [24] B. GENEST AND A. MUSCHOLL, *Pattern matching and membership for hierarchical message sequence charts*, in Proceedings of the 5th Latin American Symposium on Theoretical Informatics (LATIN 2002), Cancun, Mexico, Lecture Notes in Comput. Sci. 2286, S. Rajsbbaum, ed., Springer, Berlin, 2002, pp. 326–340.
- [25] L. M. GOLDSCHLAGER, *The monotone and planar circuit value problems are log space complete for P*, SIGACT News, 9 (1977), pp. 25–99.
- [26] L. M. GOLDSCHLAGER, *ϵ -productions in context-free grammars*, Acta Inform., 16 (1981), pp. 303–308.
- [27] G. GOTTLOB, N. LEONE, AND F. SCARCELLO, *Computing LOGCFL certificates*, Theoret. Comput. Sci., 270 (2002), pp. 761–777.
- [28] R. GREENLAW, H. J. HOOVER, AND W. L. RUZZO, *Limits to Parallel Computation: P–Completeness Theory*, Oxford University Press, New York, 1995.
- [29] C. HAGENAH, *Gleichungen mit regulären Randbedingungen über freien Gruppen*, Ph.D. thesis, University of Stuttgart, Institut für Informatik, Stuttgart, Germany, 2000.
- [30] Y. HIRSHFELD, M. JERRUM, AND F. MOLLER, *A polynomial algorithm for deciding bisimilarity of normed context-free processes*, Theoret. Comput. Sci., 158 (1996), pp. 143–159.
- [31] M. HOLZER AND K.-J. LANGE, *On the complexities of linear LL(1) and LR(1) grammars*, in Proceedings of the 9th International Symposium on Fundamentals of Computation Theory (FCT’93), Szeged, Hungary, Lecture Notes in Comput. Sci. 710, Z. Ésik, ed., Springer, Berlin, 1993, pp. 299–308.
- [32] N. IMMERMAN, *Descriptive Complexity*, Springer, New York, 1999.
- [33] M. JANTZEN, *Confluent String Rewriting*, EATCS Monogr. Theoret. Comput. Sci. 14, Springer,

- Berlin, 1988.
- [34] M. W. KRENTEL, *The complexity of optimization problems*, J. Comput. System Sci., 36 (1988), pp. 490–509.
 - [35] K.-J. LANGE AND P. MCKENZIE, *On the complexity of free monoid morphisms*, in Proceedings of the 9th International Symposium on Algorithms and Computation (ISAAC'98), Taejeon, Korea, Lecture Notes in Comput. Sci. 1533, K.-Y. Chwa and O. H. Ibarra, eds., Springer, Berlin, 1998, pp. 247–256.
 - [36] J. LARUS, *Whole program paths*, in Proceedings of the 1999 ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI), ACM, New York, 1999, pp. 259–269.
 - [37] E. LEHMAN AND A. SHELAT, *Approximation algorithms for grammar-based compression*, in Proceedings of the Thirteenth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2002), San Francisco, 2002, pp. 205–212.
 - [38] T. LENGAUER AND K. W. WAGNER, *The correlation between the complexities of the nonhierarchical and hierarchical versions of graph problems*, J. Comput. System Sci., 44 (1992), pp. 63–93.
 - [39] P. M. LEWIS II, R. E. STEARNS, AND J. HARTMANIS, *Memory bounds for recognition of context-free and context-sensitive languages*, in Proceedings of the Sixth Annual IEEE Symposium on Switching Circuit Theory and Logic Design, 1965, pp. 191–202.
 - [40] R. J. LIPTON AND Y. ZALCSTEIN, *Word problems solvable in logspace*, J. Assoc. Comput. Mach., 24 (1977), pp. 522–526.
 - [41] M. LOHREY, *Word problems and confluence problems for restricted semi-Thue systems*, in Proceedings of the 11th International Conference on Rewrite Techniques and Applications (RTA 2000), Norwich, UK, Lecture Notes in Comput. Sci. 1833, L. Bachmair, ed., Springer, Berlin, 2000, pp. 172–186.
 - [42] M. LOHREY, *Word problems for 2-homogeneous monoids and symmetric logspace*, in Proceedings of the 26th International Symposium on Mathematical Foundations of Computer Science (MFCS 2001), Mariánské Lázně, Czech Republic, Lecture Notes in Comput. Sci. 2136, J. Sgall, A. Pultr, and P. Kolman, eds. Springer, Berlin, 2001, pp. 500–511.
 - [43] M. LOHREY, *Word problems on compressed word*, in Proceedings of the 31st International Colloquium on Automata, Languages and Programming (ICALP 2004), Turku, Finland, Lecture Notes in Comput. Sci. 3142, J. Diaz, J. Karhumäki, A. Lepistö, and D. Sannella, eds., Springer, Berlin, 2004, pp. 906–918.
 - [44] M. LOHREY, *Model-checking hierarchical structures*, in Proceedings of the 20th Annual IEEE Symposium on Logic in Computer Science (LICS 2005), Chicago, IL, IEEE Computer Society Press, Los Alamitos, CA, 2005, pp. 168–177.
 - [45] M. LOHREY AND S. MANETH, *Tree automata and XPath on compressed trees*, in Proceedings of the Tenth International Conference on Implementation and Application of Automata (CIAA 2005), Sophia Antipolis, France, Lecture Notes in Comput. Sci. 3845, J. Farré, I. Litovsky, and S. Schmitz, eds., Springer, Berlin, 2006, pp. 227–240.
 - [46] S. MANETH AND G. BUSATTO, *Tree transducers and tree compressions*, in Proceedings of the 7th International Conference on Foundations of Software Science and Computation Structures (FoSSaCS 2004), Barcelona, Spain, Lecture Notes in Computer Sci. 2987, I. Walukiewicz, ed., Springer, Berlin, 2004, pp. 363–377.
 - [47] A. MARKOV, *The impossibility of certain algorithms in the theory of associative systems*, Doklady Akad. Nauk SSSR (N.S.), 58 (1947), pp. 353–356.
 - [48] P. MCKENZIE AND K. W. WAGNER, *The complexity of membership problems for circuits over sets of natural numbers*, in Proceedings of the 20th Annual Symposium on Theoretical Aspects of Computer Science (STACS 2003), Berlin, Germany, Lecture Notes in Comput. Sci. 2607, H. Alt and M. Habib, eds., Springer, Berlin, 2003, pp. 571–582.
 - [49] M. MIYAZAKI, A. SHINOHARA, AND M. TAKEDA, *An improved pattern matching algorithm for strings in terms of straight-line programs*, in Proceedings of the 8th Annual Symposium on Combinatorial Pattern Matching (CPM 97), Aarhus, Denmark, Lecture Notes in Comput. Sci., A. Apostolico and J. Hein, eds., Springer, Berlin, 1997, pp. 1–11.
 - [50] M.-J. NEDERHOF AND G. SATTÀ, *The language intersection problem for nonrecursive context-free grammars*, Inform. and Comput., 192 (2004), pp. 172–184.
 - [51] P. S. NOVIKOV, *On the Algorithmic Unsolvability of the Word Problem in Group Theory*, Amer. Math. Soc. Transl. Ser. 2 9, AMS, Providence, RI, 1958, pp. 1–122.
 - [52] C. H. PAPADIMITRIOU, *Computational Complexity*, Addison-Wesley, Reading, MA, 1994.
 - [53] C. H. PAPADIMITRIOU AND M. YANNAKAKIS, *A note on succinct representations of graphs*, Inform. and Control, 71 (1986), pp. 181–185.
 - [54] W. PLANDOWSKI, *Testing equivalence of morphisms on context-free languages*, in Proceedings of the Second Annual European Symposium on Algorithms (ESA'94), Utrecht, The Nether-

- lands, Lecture Notes in Comput. Sci. 855, J. van Leeuwen, ed., Springer, Berlin, 1994, pp. 460–470.
- [55] W. PLANDOWSKI AND W. RYTTER, *Complexity of language recognition problems for compressed words*, in *Jewels are Forever*, Contributions on Theoretical Computer Science in Honor of Arto Salomaa, J. Karhumäki, H. A. Maurer, G. Paun, and G. Rozenberg, eds., Springer, Berlin, 1999, pp. 262–272.
- [56] E. POST, *Recursive unsolvability of a problem of Thue*, J. Symbolic Logic, 12 (1947), pp. 1–11.
- [57] D. ROBINSON, *Parallel Algorithms for Group Word Problems*, Ph.D. thesis, University of California, San Diego, CA, 1993.
- [58] W. L. RUZZO, *On uniform circuit complexity*, J. Comput. System Sci., 22 (1981), pp. 365–383.
- [59] W. RYTTER, *Algorithms on compressed strings and arrays*, in Proceedings of the 26th Conference on Current Trends in Theory and Practice of Informatics (SOFSEM'99, Theory and Practice of Informatics), Milovy, Czech Republic, Lecture Notes in Comput. Sci. 1725, J. Pavelka, G. Tel, and M. Bartosek, eds., Springer, Berlin, 1999, pp. 48–65.
- [60] W. RYTTER, *Compressed and fully compressed pattern matching in one and two dimensions*, Proceedings of the IEEE, 88 (2000), pp. 1769–1778.
- [61] W. RYTTER, *Application of Lempel–Ziv factorization to the approximation of grammar-based compression*, Theoret. Comput. Sci., 302 (2003), pp. 211–222.
- [62] W. RYTTER, *Grammar compression, LZ-encodings, and string algorithms with implicit input*, in Proceedings of the 31st International Colloquium on Automata, Languages and Programming (ICALP 2004), Turku, Finland, Lecture Notes in Comput. Sci. 3142, J. Diaz, J. Karhumäki, A. Lepistö, and D. Sannella, eds., Springer, Berlin, 2004, pp. 15–27.
- [63] M. SIPSER, *Borel sets and circuit complexity*, in Proceedings of the 15th Annual Symposium on Theory of Computing (STOC 1983), ACM, New York, 1983, pp. 61–69.
- [64] L. J. STOCKMEYER, *The polynomial-time hierarchy*, Theoret. Comput. Sci., 3 (1976), pp. 1–22.
- [65] I. H. SUDBOROUGH, *On the tape complexity of deterministic context-free languages*, J. Assoc. Comput. Mach., 25 (1978), pp. 405–414.
- [66] H. VEITH, *Languages represented by Boolean formulas*, Inform. Process. Lett., 63 (1997), pp. 251–256.
- [67] H. VEITH, *How to encode a logical structure by an OBDD*, in Proceedings of the 13th Annual IEEE Conference on Computational Complexity (CoCo 1998), Buffalo, NY, IEEE Computer Society Press, Los Alamitos, CA, 1998, pp. 122–131.
- [68] H. VEITH, *Succinct representation, leaf languages, and projection reductions*, Inform. and Comput., 142 (1998), pp. 207–236.
- [69] K. W. WAGNER, *The complexity of combinatorial problems with succinct input representation*, Acta Inform., 23 (1986), pp. 325–356.
- [70] C. WRATHALL, *Complete sets and the polynomial-time hierarchy*, Theoret. Comput. Sci., 3 (1976), pp. 23–33.
- [71] K. YAMAGATA, T. UCHIDA, T. SHOUDAI, AND Y. NAKAMURA, *An effective grammar-based compression algorithm for tree structured data*, in Proceedings of the 13th International Conference on Inductive Logic Programming (ILP 2003), Szeged, Hungary, Lecture Notes in Artificial Intelligence 2835, T. Horváth, ed., Springer, Berlin, 2003, pp. 383–400.
- [72] Y. ZHANG AND R. GUPTA, *Path matching in compressed control flow traces*, in Proceedings of the 12th Data Compression Conference (DCC 2002), Snowbird, UT, IEEE Computer Society Press, Los Alamitos, CA, 2002, pp. 132–141.
- [73] J. ZIV AND A. LEMPEL, *A universal algorithm for sequential data compression*, IEEE Trans. Inform. Theory, 23 (1977), pp. 337–343.

APPROXIMATION BOUNDS FOR A GENERAL CLASS OF PRECEDENCE CONSTRAINED PARALLEL MACHINE SCHEDULING PROBLEMS*

MAURICE QUEYRANNE[†] AND ANDREAS S. SCHULZ[‡]

Abstract. An important class of scheduling problems concerns parallel machines and precedence constraints. We consider precedence delays, which associate with each precedence constraint a certain amount of time that must elapse between the completion and start times of the corresponding jobs. Together with ordinary precedence constraints, release dates and delivery times can be modeled in this manner. We present a 4-approximation algorithm for the total weighted completion time objective for this general class of problems. The algorithm is a rather simple form of list scheduling. The list is in order of job midpoints derived from a linear programming relaxation. Our analysis unifies and simplifies that of a number of special cases heretofore separately studied, while actually improving many of the former approximation results.

Key words. approximation algorithm, linear programming relaxation, performance guarantee, precedence constraints, scheduling

AMS subject classifications. 68W25, 90B35, 90C59, 68W40, 68Q25, 68M20, 06A99, 90C27

DOI. 10.1137/S0097539799358094

1. Introduction. Scheduling problems with precedence constraints are among the most difficult problems in the area of machine and processor scheduling, in particular for the design of good approximation algorithms. Our understanding of the structure of these problems and our ability to generate near-optimal solutions remain limited. The following examples illustrate this point:

(i) The first approximation algorithm for $P|\text{prec}|C_{\max}$ by Graham (1969) with performance ratio $2 - 1/m$ is still the algorithm of choice for this problem. On the other hand, it is known only that no polynomial-time algorithm can have a better approximation ratio than $4/3$, unless $P = NP$ (Lenstra and Rinnooy Kan (1978)).

(ii) The computational complexity of the problem $Pm|\text{prec}, p_j = 1|C_{\max}$, open problem “OPEN8” from the original list of Garey and Johnson (1979), is still open.

(iii) No constant-factor approximation algorithms are known for machines running at different speeds. For the makespan and total weighted completion time objectives, Chudak and Shmoys (1999) only recently improved to $O(\log m)$ performance ratios of $O(\sqrt{m})$ due to Jaffe (1980) and Schulz (1996a), respectively.

(iv) Progress is also quite recent for the latter objective on a single machine or identical parallel machines. Until recently, no constant-factor approximation algorithms were known. Lately, the use of linear programming (LP) relaxations has led to 2- and 2.7183-approximation algorithms for $1|\text{prec}|\sum w_j C_j$ and $1|r_j, \text{prec}|\sum w_j C_j$, respectively (Schulz (1996b), Schulz and Skutella (1997)), and to a 5.3281-approximation algorithm for $P|r_j, \text{prec}|\sum w_j C_j$ (Chakrabarti et al. (1996)). (Since then, Chudak

*Received by the editors June 28, 1999; accepted for publication (in revised form) June 3, 2005; published electronically March 17, 2006. An extended abstract of a preliminary version of this paper appeared in *Proceedings of the Conference on Integer Programming and Combinatorial Optimization (IPCO VI)*, Houston, TX, 1998, pp. 367–382.

<http://www.siam.org/journals/sicomp/35-5/35809.html>

[†]Sauder School of Business, University of British Columbia, 2053 Main Mall, Henry Angus 457, Vancouver, BC V6T 1Z2, Canada (Maurice.Queyranne@sauder.ubc.ca).

[‡]Sloan School of Management, Massachusetts Institute of Technology, 77 Massachusetts Avenue, E53-361, Cambridge, MA 02139-4307 (schulz@mit.edu).

and Hochbaum (1999), Chekuri and Motwani (1999), and Margot, Queyranne, and Wang (2003) have proposed various combinatorial 2-approximation algorithms for $1|\text{prec}|\sum w_j C_j$. Few deep negative results are known for these problems (Hoogeveen, Schuurman, and Woeginger (2001)).

We consider (a generalization of) the scheduling problem $P|r_j, \text{prec}|\sum w_j C_j$ and answer a question of Hall et al. (1997, page 530), which they raised in the context of a 7-approximation algorithm for this problem:

Unfortunately, we do not know how to prove a good performance guarantee for this model by using a simple list-scheduling variant.

We show that using a similar LP relaxation as Hall et al. (1997) and Chakrabarti et al. (1996) in a different way (reading the list order from the LP midpoints instead of LP completion times) yields a simple 4-approximation algorithm for $P|r_j, \text{prec}|\sum w_j C_j$. We actually obtain this result in the more general framework of precedence delays.

Let us describe the model in detail. We are given a set N of n jobs, and m identical parallel machines. Each job j has a nonnegative processing time p_j ; it must be processed uninterruptedly for that amount of time on any one of the machines. Each machine can process only one job at a time. An acyclic, directed graph $D = (N, A)$ specifies precedence constraints between jobs. A nonnegative precedence delay d_{ij} is associated with each precedence-constrained job pair $(i, j) \in A$, with the following meaning: in every feasible schedule, job j cannot start until d_{ij} time units after job i is completed. Precedence delays can be used to model ordinary precedence constraints ($d_{ij} = 0$), release dates $r_j \geq 0$ (by adding a dummy job 0 with zero processing time and precedence delays $d_{0j} = r_j$ for all other jobs), or delivery times $q_j \geq 0$, which must elapse between the end of a job's processing and its actual completion time.

Precedence delays were considered for project scheduling under the name of "finish-to-start lags," e.g., by Bartusch, Möhring, and Radermacher (1988) and Herroelen and Demeulemeester (1995), for one-machine scheduling by Wikum, Llewellyn, and Nemhauser (1994) under the name of "generalized precedence constraints," and by Balas, Lenstra, and Vazacopoulos (1995) under that of "delayed precedence constraints"; the latter authors used the L_{\max} minimization problem as a key relaxation in a modified version of the shifting bottleneck procedure for the classic job-shop scheduling problem. The one-machine problem $1|\text{prec. delays } d_{ij} = k, p_j = 1|C_{\max}$ corresponds to a basic pipeline scheduling problem; see Lawler et al. (1987) for a survey. Leung, Vornberger, and Witthoff (1984) showed that this problem is strongly NP-complete. Several other authors, including Bruno, Jones III, and So (1980), Bernstein and Gertner (1989), Palem and Simons (1993), Finta and Liu (1996), and Brucker and Knust (1999), derived polynomial-time algorithms for particular instances by utilizing well-known algorithms for special cases of the classical m -machine problem. In the context of approximation algorithms, Hall and Shmoys (1989, 1990, 1992) presented polynomial-time approximation schemes for the problems $1|r_j, q_j|C_{\max}$, $P|r_j, q_j|C_{\max}$, and $1|r_j, \text{prec. } q_j|C_{\max}$, respectively. Schuurman (1998) gave a fully polynomial-time approximation scheme for $1|\text{prec. delays } d_{ij}|C_{\max}$ when the partial order A has a special structure introduced by Wikum, Llewellyn, and Nemhauser (1994). Graham's list-scheduling algorithm (Graham (1969)) was extended to $P|\text{prec. delays } d_{ij} = k, p_j = 1|C_{\max}$ to yield a worst-case performance ratio of $2 - 1/(m(k + 1))$ (Lawler et al. (1987), Palem and Simons (1993)). This result was in turn extended by Munier to nonidentical precedence delays and processing times; see Munier, Queyranne, and Schulz (1998) for details. We refer to Brucker and Knust (1999) for an overview of complexity results for single-machine problems with precedence delays, including polynomially solvable cases with total completion time objective.

For given nonnegative job weights $w_j \geq 0$, we consider the objective of minimizing the weighted sum $\sum_{j \in N} w_j C_j$ of completion times. Here, C_j denotes the completion time of job j in the corresponding schedule. Even special cases of this problem P|prec. delays d_{ij} | $\sum w_j C_j$ are NP-hard;¹ we therefore discuss the quality of relaxations and approximation algorithms. An α -approximation algorithm is a polynomial-time algorithm that produces a solution with objective value at most α times the optimal value. Sometimes α is called the (worst-case) performance guarantee of the algorithm. Similarly, an α -relaxation is a relaxation with objective value at least $1/\alpha$ times the optimal value.

For P|prec. delays d_{ij} | $\sum w_j C_j$, we present in section 3 a new algorithm with a performance guarantee of 4. This algorithm is based on an LP relaxation of this problem, which is a direct extension of earlier LP relaxations proposed by Schulz (1996b) and Hall et al. (1997). The decision variables are the job completion times C_j ; in particular, this relaxation ignores the machine assignments. There are two sets of linear constraints: one represents the precedence delays in a straightforward fashion; the other set of constraints is a relatively simple way of enforcing the total capacity of the m machines. Although the machine assignments are ignored and the machine capacities are modeled in a simplistic way, this is sufficient to obtain the best relaxation and approximation bounds known so far for these problems and several special cases thereof. We show that using job midpoints (instead of completion times) derived from the LP relaxation leads to a performance ratio of 4 for the general problem described above. In a given schedule, the midpoint of a job is the earliest point in time at which half of its processing has been performed; if the schedule is nonpreemptive, then the midpoint of job j is simply $C_j - p_j/2$. Midpoints and more general notions of α -points have previously been used in the design and analysis of approximation algorithms for a variety of scheduling problems with the weighted sum of completion times objective; see, e.g., Phillips, Stein, and Wein (1998), Hall, Shmoys, and Wein (1996), Goemans (1997), Chekuri et al. (2001), Goemans et al. (2002), and Schulz and Skutella (2002a, 2002b).

In summary, the contributions of this paper are as follows:

(i) We shed further light on the relationship between two forms of list-scheduling algorithms: Graham's nonidling, machine-based list scheduling and job-driven list scheduling.² It is well known that the former is appropriate for optimizing objectives, such as the makespan C_{\max} , that are related to maximizing machine utilization, whereas they are inappropriate (leading to unbounded performance ratio) for job-oriented objectives, such as the weighted sum of completion times $\sum_j w_j C_j$. This difficulty was recognized by, among others, Chekuri et al. (2001), who proposed a variant of machine-based list scheduling that allows for insertion of idle time, using a mechanism for "charging" such idle time to jobs. This technique leads to a 5.8285-approximation algorithm for P| r_j , prec| $\sum w_j C_j$. We show that job-driven list-scheduling algorithms have bounded performance ratio for the $\sum_j w_j C_j$ objective if the priority list is sensibly chosen.

(ii) Using job completion times as a basis for job-driven list scheduling may yield very poor schedules for problems with parallel machines, precedence constraints, and a weighted sum of completion times objective. This may happen even if the completion

¹For example, P2| $\sum w_j C_j$, 1| r_j | $\sum w_j C_j$, and 1|prec| $\sum w_j C_j$ are NP-hard (Bruno, Coffman, Jr., and Sethi (1974), Lenstra, Rinnooy Kan, and Brucker (1977), Lawler (1978), Lenstra and Rinnooy Kan (1978)).

²Sometimes also called the parallel and serial methods; see Kolisch (1996) for a recent review in the context of resource-constrained project scheduling.

times are those of an optimal schedule. In contrast, we show that job-driven list scheduling according to job midpoints from an appropriate LP relaxation leads to job-by-job error bounds of at most 4 for a broad class of problems.

(iii) The general model of scheduling with precedence delays allows us to treat in a unified framework ordinary precedence constraints and release dates. In particular, this simplifies and unifies the analysis and proof techniques.

(iv) We present the best polynomial-time approximation bounds known so far for a broad class of parallel machine scheduling problems with precedence constraints or delays and a total weighted completion time objective. These approximation results are summarized in Table 1.1.

TABLE 1.1
Summary of results.

Problem	Performance Guarantee	Reference
P prec. delays d_{ij} $\sum w_j C_j$	4	Corollary 3.3
P prec. delays d_{ij} , $p_j = p$ $\sum w_j C_j$	3	Corollary 3.7
P r_j , prec $\sum w_j C_j$	4	Corollary 3.3
P r_j , prec, $p_j = p$ $\sum w_j C_j$	3	Corollary 3.7
P prec $\sum w_j C_j$	$4 - 2/m$	Corollary 3.5
P prec, $p_j = p$ $\sum w_j C_j$	$3 - 1/m$	Corollary 3.7
P prec = stiff $\sum w_j C_j$	$3 - 1/m$	Corollary 3.8
1 prec. delays d_{ij} $\sum w_j C_j$	3	Corollary 3.6

2. List-scheduling algorithms. List-scheduling algorithms, first analyzed by Graham (1966, 1969), are among the simplest and most commonly used approximate solution methods for parallel machine scheduling problems. These algorithms use priority rules, or job rankings. Whenever one of the m machines becomes idle, the next available job in the list is started on that machine. In the presence of precedence constraints, a job is available if all of its predecessors have completed processing. By their nonidling property, Graham's list-scheduling algorithms are appropriate when machine utilization is an important consideration. Indeed, Graham (1969) showed that list scheduling is a $(2 - 1/m)$ -approximation algorithm for P|prec| C_{\max} , no matter which priority order is used. In contrast, the two examples below show that the nonidling property may lead to poor performance ratios for a weighted sum of completion times objective $\sum_j w_j C_j$.

Example 2.1. Consider the following two-job instance of the single-machine non-preemptive scheduling problem 1| r_j | $\sum w_j C_j$ (a special case of a precedence delay problem, as discussed in the introduction). For a parameter $q \geq 2$, job 1 has $p_1 = q$, $r_1 = 0$, and $w_1 = 1$, whereas job 2 has $p_2 = 1$, $r_2 = 1$, and $w_2 = q^2$. The optimal strategy is to leave the machine idle during the time interval $[0, 1)$ so as to process job 2 first. The optimum objective value is $2q^2 + q + 2$. Any nonidling heuristic starts processing job 1 at time 0, leading to an objective value of $q^3 + q^2 + q$. The performance ratio is unbounded as q may be arbitrarily large. \square

The following example is of the same type but uses ordinary precedence constraints instead of release dates.

Example 2.2. Consider an instance with $m \geq 3$ machines and three types of jobs. Unit-time job 1 precedes jobs $a(1), a(2), \dots, a(m)$, each of which has processing time 1 as well. Jobs $b(1), b(2), \dots, b(m-1)$ are independent and have processing time m each. Job 1 and jobs $b(1), b(2), \dots, b(m-1)$ have zero weight, whereas $w_{a(h)} = 1$ for

$h = 1, 2, \dots, m$. The optimal schedule starts job 1 at time 0 on some machine and leaves $m - 1$ machines idle during the time interval $[0, 1)$ so as to complete all jobs $a(1), a(2), \dots, a(m)$ by time 2. Hence, its objective value is $2m$. Any form of Graham's machine-based list scheduling starts processing jobs $b(1), b(2), \dots, b(m - 1)$ at time 0. These jobs occupy their machines until time m , forcing jobs $a(1), a(2), \dots, a(m)$ onto the same machine as job 1, which results in an objective value of $(m + 1)(m + 2)/2 - 1$. Therefore, the performance ratio increases with m . \square

Evidently, the appropriate introduction of idle time is an important element in the design of approximation algorithms to minimize a weighted sum of completion times subject to precedence delays. As Examples 2.1 and 2.2 illustrate, idle time is needed to prevent large-weight jobs, which may soon become available, from being delayed by other, less important jobs.³ On the other hand, too much idle time is undesirable as well. The necessity to balance these two effects contributes to the difficulty of this problem. All former approximation algorithms for $P|r_j, \text{prec}|\sum w_j C_j$ with constant-factor performance ratios are based on variants of Graham's original list scheduling, which actually tries to avoid machine idle time. In fact, Hall et al. (1997) partitioned jobs into groups that are individually scheduled according to Graham's list-scheduling rule, and then these schedules are concatenated to obtain a solution for the original problem. To find a good partition, this scheme was enriched with randomness by Chakrabarti et al. (1996). Chekuri et al. (2001) presented a different variant of Graham's list scheduling by artificially introducing idle time whenever it seems that a further delay of the next available job in the list (if it is not the first) can be afforded. It is worth mentioning that these techniques, analyses, and approximation results also generalize to precedence delays.

Another, arguably simpler, strategy is to consider the jobs one by one, in the given list order, starting from an empty schedule. Each job is nonpreemptively inserted into the current schedule without altering the jobs already scheduled. Specific job-driven list-scheduling algorithms differ in how this principle is implemented. For definiteness, consider the version in Figure 2.1, whereby every job is considered in the list order and is scheduled at the earliest feasible time at the end of the current schedule on a machine. We assume that the given list is a linear extension of the partial order defined by the precedence constraints.

1. The list $L = (\ell(1), \ell(2), \dots, \ell(n))$ is given.
2. Initially, all machines are empty, with machine completion times $\Gamma_h := 0$ for $h = 1, 2, \dots, m$.
3. For $k = 1$ to n do:
 - 3.1 Let job $j = \ell(k)$; set its start time $S_j := \max(\max\{C_i + d_{ij} : (i, j) \in A\}, \min\{\Gamma_h : h = 1, 2, \dots, m\})$ and its completion time $C_j := S_j + p_j$.
 - 3.2 Assign job j to a machine h such that $\Gamma_h \leq S_j$. Update $\Gamma_h := C_j$.

FIG. 2.1. *Job-driven list-scheduling algorithm for $P|\text{prec. delays } d_{ij}|\sum w_j C_j$.*

Various rules may be used in step 3.2 for the choice of the assigned machine h , for example, one with largest completion time Γ_h (so as to reduce the idle time between Γ_h and S_j). Moreover, the above algorithm can be modified to allow the insertion of

³It is important to point out that this difficulty results from the nonpreemptive mode; Graham's list scheduling with jobs ordered according to LP completion times gives a 3-approximation for $P|r_j, \text{prec, pmtn}|\sum w_j C_j$ (Hall et al. (1997)).

a job in an idle period before time Γ_h . In effect, the observations below also apply to all these variants.

One method (e.g., Phillips, Stein, and Wein (1998) and Hall et al. (1997)) for defining the list L consists of sorting the jobs in nondecreasing order of their completion times in a relaxation of the scheduling problem under consideration. In the presence of ordinary precedence constraints, this works well for the case of a single machine (Schulz (1996b), Hall et al. (1997)), but Example 2.3 shows that this may produce very poor schedules for the case of identical parallel machines. This example uses the list which is produced by an optimal schedule, the tightest kind of relaxation that can be defined; moreover, this optimal schedule defines the same completion time order as the LP relaxation in section 3.1.

Example 2.3. For a fixed number $m \geq 2$ of identical parallel machines and a positive number ϵ , let the job set N consist of m sets J_h ($h = 1, 2, \dots, m$) of $m + 1$ jobs each, plus a “last job” n . (Thus $n = m(m + 1) + 1$.) Each set J_h consists of a “long job” $a(h)$, with processing time $p_{a(h)} = 1 + (h - 1)(m + 1)\epsilon$, and m “small jobs” $b(h, g)$ (for $g = 1, 2, \dots, m$), each with processing time $p_{b(h, g)} = \epsilon$ and subject to the precedence constraint $(a(h), b(h, g))$. In addition, there is a precedence constraint $(b(h, g), n)$ from each small job $b(h, g)$ (for $h = 1, 2, \dots, m$ and $g = 1, 2, \dots, m$) to the last job n , which has processing time ϵ . The objective is to minimize either the makespan or a weighted sum $\sum_j w_j C_j$ of job completion times with weights $w_j = 0$ for all $j \neq n$ and $w_n = 1$; due to the precedence constraints, these two objectives coincide for any feasible schedule. An optimal schedule has, for $h = 1, 2, \dots, m$, long job $a(h)$ starting at time $S_{a(h)}^* = 0$ on machine h , immediately followed by all small jobs $b(h, g)$ (for $g = 1, 2, \dots, m$) in the same set J_h , assigned as uniformly as possible to machines $1, 2, \dots, h$. Note that all the jobs in J_h are completed before the next long job $a(h + 1)$ completes. Job n is then processed last on any machine, so that the optimal objective value is $C_{\max}^* = C_n^* = 1 + (m^2 + 1)\epsilon$. On the other hand, any version of the job-driven list-scheduling algorithm with all jobs listed in order of their optimal completion times produces the following schedule: long job $a(1)$ is scheduled first, with start time $S_{a(1)}^L = 0$ and completion time $C_{a(1)}^L = 1$; the m small jobs $b(1, g)$ (for $g = 1, 2, \dots, m$) in J_1 are then scheduled, each with start time $S_{b(1, g)}^L = 1$ and completion time $C_{b(1, g)}^L = 1 + \epsilon$ on a different machine; this forces all subsequent jobs to be scheduled no earlier than date $1 + \epsilon$. Consequently, for $h = 1, 2, \dots, m$, long job $a(h)$ is scheduled with start time $S_{a(h)}^L = (h - 1) + (h - 1)(h - 1)(m + 1)/2\epsilon$, followed by all small jobs $b(h, g)$ (for $g = 1, 2, \dots, m$) in the same set J_h , each with start time $S_{b(h, g)}^L = S_{a(h)}^L + p_{a(h)}$ on a different machine. Finally, job n is scheduled last with $S_n^L = m + (m + m(m - 1)(m + 1)/2)\epsilon$, and thus the objective value is $C_{\max}^L = C_n^L = m + o(\epsilon)$, arbitrarily close to m times the optimal value C_{\max}^* when $\epsilon > 0$ is small enough. \square

3. LP-based approximation algorithms. In this section we present an LP relaxation for the problem of minimizing a weighted sum $\sum_j w_j C_j$ of job completion times subject to precedence delays, and use it to develop a 4-approximation algorithm for this problem. Thereafter, we refine the analysis to give improved bounds for various relevant special cases.

3.1. The LP relaxation. The decision variables are the job completion times C_j . The set of constraints is

$$(3.1) \quad C_j \geq p_j \quad \text{for all } j \in N,$$

$$(3.2) \quad C_j \geq C_i + d_{ij} + p_j \quad \text{for all } (i, j) \in A,$$

$$(3.3) \quad \sum_{j \in F} p_j C_j \geq \frac{1}{2m} \left(\sum_{j \in F} p_j \right)^2 + \frac{1}{2} \sum_{j \in F} p_j^2 \quad \text{for all } F \subseteq N.$$

This relaxation is an extension of a relaxation introduced by Hall et al. (1997) for $P|r_j, \text{prec}|\sum w_j C_j$: on the one hand, ordinary precedence constraints are replaced by inequalities (3.2), which model the precedence delays; on the other hand, inequalities (3.3) are stronger than the analogous class of inequalities used by Hall et al. (1997). Our analysis requires this strengthened class of inequalities, which was proposed by Schulz (1996b). For $m = 1$ (the single-machine case), they are identical to inequalities introduced by Wolsey (1985) to model the constraint that the machine can process at most one job at a time; see Queyranne (1993) for further details. Constraints (3.1) impose the trivial lower bounds on job completion times.

For a weighted sum of completion times objective, the LP formulation is simply

$$(3.4) \quad \text{minimize } \sum_{j \in N} w_j C_j \quad \text{subject to (3.1)–(3.3).}$$

Although there is an exponential number of constraints in (3.3), the separation problem for these inequalities can be solved in polynomial time (Schulz 1996a). It follows that this LP relaxation can be solved in polynomial time, using the ellipsoid method.

3.2. The approximation algorithm. Let C^{LP} denote any feasible solution to the constraint set (3.1)–(3.3) of this LP. We use this LP solution to define a feasible schedule with completion time vector C^{H} and analyze the job-by-job relationship between C_j^{H} and C_j^{LP} for every job $j \in N$. We define the LP midpoint as $M_j^{\text{LP}} := C_j^{\text{LP}} - p_j/2$. We now use the list-scheduling algorithm of Figure 2.1 with the LP midpoint list L defined by sorting the jobs in nondecreasing order of their midpoints M_j^{LP} . The next theorem contains our main result.

THEOREM 3.1. *Let C^{LP} denote any feasible solution to the constraint set (3.1)–(3.3), and let M^{LP} denote the corresponding vector of LP midpoints. Let S^{H} be the vector of start times of the feasible schedule constructed by the job-driven list-scheduling algorithm using the LP midpoint list. Then*

$$(3.5) \quad S_j^{\text{H}} \leq 4 M_j^{\text{LP}} \quad \text{for all jobs } j \in N.$$

Proof. Assume for simplicity that the jobs are indexed in the order of their LP midpoints; that is, $M_1^{\text{LP}} \leq M_2^{\text{LP}} \leq \dots \leq M_n^{\text{LP}}$. We fix job $j \in N$ and consider the schedule constructed by the list-scheduling heuristic using the LP midpoint list $L = (1, 2, \dots, n)$ up to and including the scheduling of job j , that is, up to the completion of step 3 with $k = j$. Let $[j] := \{1, 2, \dots, j\}$.

Let μ denote the total time between 0 and the start time S_j^{H} of job j when all m machines are busy at this stage of the algorithm. Since only jobs in $[j - 1]$ have been scheduled so far, $\mu \leq \sum_{i=1}^{j-1} p_i/m$. Let $\lambda := S_j^{\text{H}} - \mu$. To prove (3.5), we need only show that

$$(i) \quad \frac{1}{m} \sum_{i=1}^{j-1} p_i \leq 2M_j^{\text{LP}} \quad \text{and} \quad (ii) \quad \lambda \leq 2M_j^{\text{LP}}.$$

Inequality (i) follows from a straightforward variant of Lemma 1 in Schulz (1996b) or Lemma 3.2 in Hall et al. (1997). For this, first observe that the inequalities (3.3) are

equivalent to

$$\sum_{i \in F} p_i M_i \geq \frac{1}{2m} \left(\sum_{i \in F} p_i \right)^2 \quad \text{for all } F \subseteq N.$$

Since M^{LP} satisfies all these inequalities, letting $F = [j-1]$ and using $M_1^{\text{LP}} \leq M_2^{\text{LP}} \leq \dots \leq M_j^{\text{LP}}$, we obtain

$$\left(\sum_{i=1}^{j-1} p_i \right) M_j^{\text{LP}} \geq \sum_{i=1}^{j-1} p_i M_i^{\text{LP}} \geq \frac{1}{2m} \left(\sum_{i=1}^{j-1} p_i \right)^2,$$

implying (i).

To show (ii), let q denote the number of maximal intervals between dates 0 and S_j^{H} when at least one machine is idle (i.e., not processing a job in $[j-1]$) in the schedule C^{H} . Denote these idle intervals as $[a_h, b_h)$ for $h = 1, 2, \dots, q$, so that $0 \leq a_1, b_{h-1} < a_h < b_h$ for all $h = 2, \dots, q$, and $b_q \leq S_j^{\text{H}}$. Hence, $\lambda = \sum_{h=1}^q (b_h - a_h)$ and all machines are busy during the complementary intervals $[b_h, a_{h+1})$, including intervals $[0, a_1)$ and $[b_q, S_j^{\text{H}})$, if nonempty.

Consider the digraph $G^{[j]} = ([j], A^{[j]})$, where

$$A^{[j]} := \{(k, \ell) \in A : k, \ell \in [j] \text{ and } C_\ell^{\text{H}} = C_k^{\text{H}} + d_{k\ell} + p_\ell\}.$$

That is, $A^{[j]}$ is the set of precedence pairs in $[j]$ for which the precedence delay constraints (3.2) are tight for C^{H} . If $b_q > 0$, then a machine becomes busy at date b_q (or starts processing job j if $b_q = S_j^{\text{H}}$) and thus there exists a job $x(q) \in [j]$ with start time $S_{x(q)}^{\text{H}} = b_q$. Since $x(q) \in [j]$ we have $M_{x(q)}^{\text{LP}} \leq M_j^{\text{LP}}$. We repeat the following process for decreasing values of the interval index h , starting with $h = q$, until we reach the date 0 or the busy interval $[0, a_1)$. Let $(v(1), \dots, v(s))$ denote a maximal path in $G^{[j]}$ with last node (job) $v(s) = x(h)$. Note that we must have $b_g < S_{v(1)}^{\text{H}} \leq a_{g+1}$ for some busy interval $[b_g, a_{g+1})$ with $a_{g+1} < b_h$, for otherwise some machine is idle immediately before the start time $S_{v(1)}^{\text{H}}$ of job $v(1)$ and this job, not being constrained by any tight precedence delay constraint, should have started earlier than that date. (Unless $S_{v(1)}^{\text{H}} = 0$, of course. In this case, a_{g+1} is the first point in time at which some machine falls idle.) We have

$$(3.6) \quad b_h - a_{g+1} \leq S_{v(s)}^{\text{H}} - S_{v(1)}^{\text{H}} = \sum_{i=1}^{s-1} (S_{v(i+1)}^{\text{H}} - S_{v(i)}^{\text{H}}) = \sum_{i=1}^{s-1} (p_{v(i)} + d_{v(i)v(i+1)}).$$

On the other hand, the precedence delay constraints (3.2) imply

$$M_{v(i+1)}^{\text{LP}} \geq M_{v(i)}^{\text{LP}} + \frac{1}{2} p_{v(i)} + d_{v(i)v(i+1)} + \frac{1}{2} p_{v(i+1)}$$

for all $i = 1, 2, \dots, s-1$. Therefore,

$$M_{x(h)}^{\text{LP}} - M_{v(1)}^{\text{LP}} \geq \frac{1}{2} \sum_{i=1}^{s-1} (p_{v(i)} + d_{v(i)v(i+1)}) \geq \frac{1}{2} (b_h - a_{g+1}).$$

If $b_g > 0$, then let $x(g)$ be a job with $S_{x(g)}^{\text{H}} = b_g$. Because of the order of jobs in the priority list L , $S_{x(g)}^{\text{H}} < S_{v(1)}^{\text{H}}$ implies $M_{x(g)}^{\text{LP}} \leq M_{v(1)}^{\text{LP}}$. Therefore,

$$(3.7) \quad M_{x(h)}^{\text{LP}} - M_{x(g)}^{\text{LP}} \geq \frac{1}{2} (b_h - a_{g+1}).$$

We also have $(k, x(g)) \in A^{[j]}$ for some $k \in [j]$ with $S_k^H < b_g = S_{x(g)}^H$, for otherwise job $x(g)$ should have started processing on some idle machine before date b_g . We may thus repeat the above process with $h = g$ and job $x(h) = x(g)$. Since $g < h$ at each step, this whole process must terminate, generating a decreasing sequence of indices $q = H(1) > \dots > H(q') = 0$ such that every idle interval is contained in some interval $[a_{H(i+1)+1}, b_{H(i)}]$. Adding the corresponding inequalities (3.7), we obtain

$$(3.8) \quad \lambda \leq \sum_{i=1}^{q'-1} (b_{H(i)} - a_{H(i+1)+1}) \leq 2(M_{x(H(1))}^{LP} - M_{x(H(q'))}^{LP}) \leq 2(M_j^{LP} - 0).$$

This establishes (ii). The proof of Theorem 3.1 is complete. \square

The following example shows that the factor 4 in inequality (3.5) is (asymptotically) best possible, even for ordinary precedence constraints only.

Example 3.2. For a given number $m \geq 2$ of identical parallel machines, the job set N includes m sets J_h (for $h = 1, 2, \dots, m$) of $m + 1$ jobs each: a job $a(h)$ with processing time $p_{a(h)} = 2^{h-1-m}$, and m “small jobs” $b(h, g)$ (for $g = 1, 2, \dots, m$), each with processing time $p_{b(h,g)} = 0$ and subject to the ordinary precedence constraint $(a(h), b(h, g))$ (with zero delay). In addition, there are m “unit jobs” $u(i)$ (for $i = 1, 2, \dots, m$) with processing time $p_{u(i)} = 1$, and two jobs, $n - 1$ and n , with processing times $p_{n-1} = 2/m$ and $p_n = 0$ (thus $n = (m + 1)^2 + 1$). There are two additional (ordinary) precedence constraints: $a(m)$ precedes $n - 1$ and $n - 1$ precedes n .

For sufficiently large m , the following solution C^{LP} is feasible for constraints (3.1)–(3.3): $C_j^{LP} = p_{a(h)} = 2^{h-1-m}$ for all jobs $j \in J_h$, for $h = 1, 2, \dots, m$; $C_{u(i)}^{LP} = 1 + 2/m$ for all unit jobs $u(i)$; and $C_{n-1}^{LP} = C_n^{LP} = M_n^{LP} = p_{a(m)} + p_{n-1} = 1/2 + 2/m$. Therefore, an LP midpoint list starts with sets J_1, J_2, \dots, J_m (each with its medium job before all its small jobs), followed by job $n - 1$, all unit jobs $u(i)$, and finally job n . This LP midpoint list produces the following schedule. First schedule the sets J_h in sequence $h = 1, 2, \dots, m$, beginning with the medium job $a(h)$ on one machine (starting just after set $J(h - 1)$ is complete), immediately followed by the m small jobs in J_h , each on a different machine. All jobs j in J_h have completion time $C_j^H = \sum_{i=1}^h p_{a(i)} = 2^{h-m} - 2^{-m}$; since all machines are occupied at that date, this forces all subsequent jobs to start no earlier than that date. After the last set J_m is complete, schedule job $n - 1$ and $m - 1$ unit jobs $u(i)$, each on a different machine. After this, start the remaining unit job on the same machine as job $n - 1$. The first $m - 1$ unit jobs have completion time $C_{u(i)}^H = 2 - 2^{-m}$. Finally, start job n at date $S_n^H = 2 - 2^{-m}$. For m large enough, the latter expression is arbitrarily close to $4M_n^{LP} = 2 + 8/m$. \square

If C^{LP} is an optimal LP solution, Theorem 3.1 implies performance ratios of $1/4$ and 4 for the LP relaxation and the heuristic solution, respectively.

COROLLARY 3.3. *Let C^{LP} denote an optimal solution to the LP relaxation defined in (3.4) for the problem $P|\text{prec. delays } d_{ij}|\sum w_j C_j$. Let C^H denote the solution constructed from C^{LP} by the job-driven list-scheduling algorithm using the LP midpoint list, and let C^* denote an optimum schedule. Then,*

$$(3.9) \quad \sum_{j \in N} w_j C_j^{LP} \geq \frac{1}{4} \sum_{j \in N} w_j C_j^* \quad \text{and} \quad \sum_{j \in N} w_j C_j^H \leq 4 \sum_{j \in N} w_j C_j^*.$$

The following example shows that the latter bound is (asymptotically) tight, even for ordinary precedence constraints only, i.e., for problem $P|\text{prec. delays } d_{ij}|\sum w_j C_j$.

Example 3.4. In Example 3.2, let the weights be $w_n = 1$ and all other $w_j = 0$, so that the optimum solution has $\sum_{j \in N} w_j C_j^* = w_n(p_{a(m)} + p_{n-1} + p_n) = 1/2 + 2/m$. Since the precedence constraints (3.2) and the lower bounds (3.1) imply $C_n^{\text{LP}} \geq p_{a(m)} + p_{n-1} + p_n = 1/2 + 2/m$, the solution C^{LP} described in Example 3.2 is optimal for the LP relaxation (3.4); its objective value is $\sum_{j \in N} w_j C_j^{\text{LP}} = 1/2 + 2/m = \sum_{j \in N} w_j C_j^*$. Using the LP midpoint list produces the schedule described in the above example, with $\sum_{j \in N} w_j C_j^{\text{H}} = 2 - 2^{-m}$. For m large enough, the latter expression is arbitrarily close to $4 \sum_{j \in N} w_j C_j^*$. \square

We suspect that the first inequality in (3.9), bounding the performance ratio of the LP relaxation, is not tight. The worst instances we know arise actually for $m = 1$ and lead to a gap of 2; see Hall et al. (1997) for details.

3.3. Special cases. The analysis in Theorem 3.1 can be refined for some special cases, yielding tighter performance ratios. For the problem $\text{P|prec|} \sum w_j C_j$, observe that the list-scheduling algorithm will not allow all machines to be simultaneously idle at any date before the start time of any job $j \in N$. Therefore, in the proof of Theorem 3.1, all the idle intervals, with total length λ , contain some processing of some job(s) $i < j$; as a result the total work during the busy intervals is at most $\sum_{i=1}^{j-1} p_i - \lambda$. Hence, we obtain the following result.

COROLLARY 3.5. *Job-driven list scheduling by optimal LP midpoints is a $(4 - 2/m)$ -approximation algorithm for the scheduling problem $\text{P|prec|} \sum w_j C_j$.*

Note that for $m = 1$ we recover the performance ratio of 2 for $1|\text{prec}| \sum w_j C_j$ in Schulz (1996b), which is known to be tight for that special case (Hall et al. (1997)).

In the case of precedence delays and a single machine, the idle intervals that add up to λ time units cannot contain any processing. Therefore, in the proof of Theorem 3.1 replace inequality (3.6) with $b_h - a_{g+1} \leq S_{v(s)}^{\text{H}} - C_{v(1)}^{\text{H}} = \sum_{i=1}^{s-1} d_{v(i)v(i+1)}$. (Note that, if all processing times are positive, then $s = 2$ and the summation in the right-hand side consists of a single term $d_{v(1)v(2)}$.) Adding up the precedence delay constraints (3.2) for all $i = 1, 2, \dots, s - 1$ and omitting some processing times yields $M_{x(h)}^{\text{LP}} - M_{v(1)}^{\text{LP}} \geq \sum_{i=1}^{s-1} d_{v(i)v(i+1)} \geq b_h - a_{g+1}$. Therefore, we may replace (3.8) with $\lambda \leq \sum_{i=1}^{q'-1} (b_{H(i)} - a_{H(i+1)+1}) \leq M_{x(H(1))}^{\text{LP}} - M_{x(H(q'))}^{\text{LP}} \leq M_j^{\text{LP}}$ and thus inequality (ii) with $\lambda \leq M_j^{\text{LP}}$. This implies $S_j^{\text{H}} \leq 3M_j^{\text{LP}}$.

COROLLARY 3.6. *Job-driven list scheduling by optimal LP midpoints is a 3-approximation algorithm for the scheduling problem $1|\text{prec. delays } d_{ij}| \sum w_j C_j$.*

Note that for the special case $1|r_j, \text{prec}| \sum w_j C_j$ we recover the performance ratio of 3 in Schulz (1996b). The best approximation algorithm known for this problem, however, has a performance guarantee of $e \approx 2.7183$ (Schulz and Skutella (1997)).

In principle, we may use any LP α -point $C_j^{\text{LP}}(\alpha) := C_j^{\text{LP}} - (1 - \alpha)p_j$ for some $0 \leq \alpha < 1$ in section 3.2. Indeed, inequality (i) in the proof of Theorem 3.1 can be replaced with $\sum_{i=1}^{j-1} p_i/m \leq 2C_j^{\text{LP}}(\alpha)$, provided that $\alpha \geq 1/2$. On the other hand, inequality (ii) becomes $\lambda \leq (1 - \alpha)^{-1}C_j^{\text{LP}}(\alpha)$. While it turns out that using the midpoint $M_j^{\text{LP}} = C_j^{\text{LP}}(1/2)$ leads to the best bound for the general case, we can take advantage of this observation for some special cases.

For example, if the LP midpoint list coincides with the LP start-time list (which is the case, e.g., if $p_i = p_j$ for all $i, j \in N$), then we can apply inequality (i) in the proof of Theorem 3.1 with $\alpha = 1/2$ to bound the total busy time, whereas we can use inequality (ii) with $\alpha = 0$ to bound the total idle time λ by $S_j^{\text{LP}} = C_j^{\text{LP}}(0)$. We obtain the following result.

COROLLARY 3.7. Let C^{LP} denote an optimal solution to the LP relaxation (3.4). If $M_1^{\text{LP}} \leq M_2^{\text{LP}} \leq \dots \leq M_n^{\text{LP}}$ implies $S_1^{\text{LP}} \leq S_2^{\text{LP}} \leq \dots \leq S_n^{\text{LP}}$, then job-driven list scheduling by LP midpoints is a 3-approximation algorithm for the problem $P|\text{prec. delays } d_{ij}|\sum w_j C_j$.

Because the arguments leading to Corollary 3.5 apply here as well, Corollary 3.7 actually gives a $(3-1/m)$ -approximation algorithm for this special case of $P|\text{prec}|\sum w_j C_j$. Hall et al. (1997) had earlier proved these performance ratios for the special cases $P|r_j, \text{prec}, p_j = 1|\sum w_j C_j$ and $P|\text{prec}, p_j = 1|\sum w_j C_j$, respectively.

Let us finally consider “stiff” instances of $P|\text{prec}|\sum w_j C_j$. Margot, Queyranne, and Wang (2003) called an instance (of the single-machine problem $1|\text{prec}|\sum w_j C_j$) stiff if $w(I)/p(I) \leq w(N)/p(N)$ for all initial sets $I \subseteq N$. A set I is initial if $j \in I$ and $(i, j) \in A$ imply $i \in I$. Chekuri and Motwani (1999) and Margot, Queyranne, and Wang (2003) showed that the total weighted completion time of any feasible one-machine schedule of a stiff instance is within a factor 2 of that of an optimum. Given an instance of $P|\text{prec}|\sum w_j C_j$, we define a corresponding single-machine instance with processing times p_j/m , and we keep the original job weights and precedence constraints. The objective value of an optimal solution to this single-machine instance is a lower bound on the cost of an optimal schedule C^* for the original instance on m identical parallel machines (Chekuri et al. (2001)). Let us argue that job-driven list scheduling according to optimal LP start times $S_j^{\text{LP}} = C_j^{\text{LP}} - p_j$ is a $(3 - 1/m)$ -approximation for stiff instances of $P|\text{prec}|\sum w_j C_j$. In fact, it suffices when C^{LP} is an optimal solution to the LP with constraint set (3.1)–(3.2), which can be solved combinatorially using shortest-path computations. That is, inequalities (3.3) are not needed. So, let L be defined by $S_1^{\text{LP}} \leq S_2^{\text{LP}} \leq \dots \leq S_n^{\text{LP}}$. From the preceding discussion, we already know that we can use inequality (ii) in the proof of Theorem 3.1 with $\alpha = 0$ to bound the total idle time λ by S_j^{LP} . It remains to bound $\sum_{i=1}^j p_i/m$. This time, we do not give a job-by-job bound, but bound the entire contribution of busy periods to the objective function value of the heuristic schedule. Let C^1 be the completion time vector of an optimal schedule to the corresponding single-machine instance. Note that $\sum_{i=1}^j p_i/m$ is the completion time of job j in the single-machine schedule where jobs are sequenced according to L . Hence, as the instance is stiff, $\sum_{j=1}^n w_j \sum_{i=1}^j p_i/m \leq 2 \sum_{j=1}^n w_j C_j^1$. Overall, we obtain

$$\begin{aligned} \sum_{j=1}^n w_j C_j^{\text{H}} &\leq \sum_{j=1}^n w_j \left(\sum_{i=1}^j p_i/m + (1 - 1/m)(S_j^{\text{LP}} + p_j) \right) \\ &\leq 2 \sum_{j=1}^n w_j C_j^1 + (1 - 1/m) \sum_{j=1}^n w_j C_j^{\text{LP}} \\ &\leq (3 - 1/m) \sum_{j=1}^n w_j C_j^*. \end{aligned}$$

COROLLARY 3.8. Let C^{LP} denote an optimal solution to the LP relaxation defined over (3.1)–(3.2). Job-driven list scheduling by LP start times is a combinatorial $(3 - 1/m)$ -approximation algorithm for stiff instances of the scheduling problem $P|\text{prec}|\sum w_j C_j$.

Finally, let us point out that Corollaries 3.5, 3.6, and 3.7 also imply corresponding bounds on the quality of the LP relaxation (3.4) for these special cases.

Acknowledgments. The authors are grateful to Alix Munier for helpful discussions and to Maxim Sviridenko for useful comments on an earlier version of this paper. They also thank the organizers of the workshop on “Parallel Scheduling” held July 14–18, 1997, at Schloß Dagstuhl, Germany, where this work was initiated.

REFERENCES

- E. BALAS, J. K. LENSTRA, AND A. VAZACOPOULOS (1995), *The one-machine problem with delayed precedence constraints and its use in job shop scheduling*, *Manag. Sci.*, 41, pp. 94–109.
- M. BARTUSCH, R. H. MÖHRING, AND F. J. RADERMACHER (1988), *Scheduling project networks with resource constraints and time windows*, *Ann. Oper. Res.*, 16, pp. 201–240.
- D. BERNSTEIN AND I. GERTNER (1989), *Scheduling expressions on a pipelined processor with a maximal delay of one cycle*, *ACM Trans. Program. Lang. Syst.*, 11, pp. 57–66.
- P. BRUCKER AND S. KNUST (1999), *Complexity results for single-machine problems with positive finish-start time-lags*, *Computing*, 63, pp. 299–316.
- J. L. BRUNO, J. W. JONES III, AND K. SO (1980), *Deterministic scheduling with pipelined processors*, *IEEE Trans. Comput.*, 29, pp. 308–316.
- L. BRUNO, E. G. COFFMAN, JR., AND R. SETHI (1974), *Scheduling independent tasks to reduce mean finishing time*, *Comm. ACM*, 17, pp. 382–387.
- S. CHAKRABARTI, C. A. PHILLIPS, A. S. SCHULZ, D. B. SHMOYS, C. STEIN, AND J. WEIN (1996), *Improved scheduling algorithms for minsum criteria*, in *Automata, Languages and Programming*, Lecture Notes in Comput. Sci. 1099, F. Meyer auf der Heide and B. Monien, eds., Springer, Berlin, pp. 646–657.
- C. CHEKURI AND R. MOTWANI (1999), *Precedence constrained scheduling to minimize sum of weighted completion time on a single machine*, *Discrete Appl. Math.*, 98, pp. 29–38.
- C. CHEKURI, R. MOTWANI, B. NATARAJAN, AND C. STEIN (2001), *Approximation techniques for average completion time scheduling*, *SIAM J. Comput.*, 31, pp. 146–166.
- F. A. CHUDAK AND D. S. HOCHBAUM (1999), *A half-integral linear programming relaxation for scheduling precedence-constrained jobs on a single machine*, *Oper. Res. Lett.*, 25, pp. 199–204.
- F. A. CHUDAK AND D. B. SHMOYS (1999), *Approximation algorithms for precedence-constrained scheduling problems on parallel machines that run at different speeds*, *J. Algorithms*, 30, pp. 323–343.
- L. FINTA AND Z. LIU (1996), *Single machine scheduling subject to precedence delays*, *Discrete Appl. Math.*, 70, pp. 247–266.
- M. R. GAREY AND D. S. JOHNSON (1979), *Computers and Intractability: A Guide to the Theory of NP-Completeness*, W. H. Freeman, San Francisco.
- M. X. GOEMANS (1997), *Improved approximation algorithms for scheduling with release dates*, in *Proceedings of the 8th Annual ACM-SIAM Symposium on Discrete Algorithms*, New Orleans, LA, pp. 591–598.
- M. X. GOEMANS, M. QUEYRANNE, A. S. SCHULZ, M. SKUTELLA, AND Y. WANG (2002), *Single machine scheduling with release dates*, *SIAM J. Discrete Math.*, 15, pp. 165–192.
- R. L. GRAHAM (1966), *Bounds for certain multiprocessing anomalies*, *Bell System Tech. J.*, 45, pp. 1563–1581.
- R. L. GRAHAM (1969), *Bounds on multiprocessing timing anomalies*, *SIAM J. Appl. Math.*, 17, pp. 416–429.
- L. A. HALL, A. S. SCHULZ, D. B. SHMOYS, AND J. WEIN (1997), *Scheduling to minimize average completion time: Off-line and on-line approximation algorithms*, *Math. Oper. Res.*, 22, pp. 513–544.
- L. A. HALL AND D. B. SHMOYS (1989), *Approximation schemes for constrained scheduling problems*, in *Proceedings of the 30th Annual IEEE Symposium on Foundations of Computer Science*, Research Triangle Park, NC, pp. 134–139.
- L. A. HALL AND D. B. SHMOYS (1990), *Near-optimal sequencing with precedence constraints*, in *Proceedings of the 1st Integer Programming and Combinatorial Optimization Conference*, R. Kannan and W. R. Pulleyblank, eds., University of Waterloo Press, Waterloo, ON, pp. 249–260.
- L. A. HALL AND D. B. SHMOYS (1992), *Jackson’s rule for single-machine scheduling: Making a good heuristic better*, *Math. Oper. Res.*, 17, pp. 22–35.
- L. A. HALL, D. B. SHMOYS, AND J. WEIN (1996), *Scheduling to minimize average completion time: Off-line and on-line algorithms*, in *Proceedings of the 7th Annual ACM-SIAM Symposium on Discrete Algorithms*, Atlanta, GA, pp. 142–151.
- W. S. HERROELEN AND E. L. DEMEULEMEESTER (1995), *Recent advances in branch-and-bound procedures for resource-constrained project scheduling problems*, in *Scheduling Theory and Its Applications*, P. Chrétienne, E. G. Coffman, Jr., J. K. Lenstra, and Z. Liu, eds., John Wiley and Sons, Chichester, UK, pp. 259–276.

- J. A. HOOGEVEEN, P. SCHURMAN, AND G. J. WOEINGER (2001), *Non-approximability results for scheduling problems with minsum criteria*, *INFORMS J. Comput.*, 13, pp. 157–168.
- J. M. JAFFE (1980), *Efficient scheduling of tasks without full use of processor resources*, *Theoret. Comput. Sci.*, 12, pp. 1–17.
- R. KOLISCH (1996), *Serial and parallel resource-constrained project scheduling methods revisited: Theory and computation*, *European J. Oper. Res.*, 90, pp. 320–333.
- E. LAWLER, J. K. LENSTRA, C. MARTEL, B. SIMONS, AND L. STOCKMEYER (1987), *Pipeline Scheduling: A Survey*, Technical report RJ 5738 (57717), IBM Research Division, San Jose, CA.
- E. L. LAWLER (1978), *Sequencing jobs to minimize total weighted completion time subject to precedence constraints*, *Ann. Discrete Math.*, 2, pp. 75–90.
- J. K. LENSTRA AND A. H. G. RINNOOY KAN (1978), *Complexity of scheduling under precedence constraints*, *Oper. Res.*, 26, pp. 22–35.
- J. K. LENSTRA, A. H. G. RINNOOY KAN, AND P. BRUCKER (1977), *Complexity of machine scheduling problems*, *Ann. Discrete Math.*, 1, pp. 343–362.
- J. Y.-T. LEUNG, O. VORNBERGER, AND J. D. WITTHOFF (1984), *On some variants of the bandwidth minimization problem*, *SIAM J. Comput.*, 13, pp. 650–667.
- F. MARGOT, M. QUEYRANNE, AND Y. WANG (2003), *Decompositions, network flows, and a precedence constrained single-machine scheduling problem*, *Oper. Res.*, 51, pp. 981–992.
- A. MUNIER, M. QUEYRANNE, AND A. S. SCHULZ (1998), *Approximation bounds for a general class of precedence constrained parallel machine scheduling problems*, in *Integer Programming and Combinatorial Optimization*, Lecture Notes in Comput. Sci. 1412, R. Bixby, E. A. Boyd, and R. Z. Rios Mercado, eds., Springer, Berlin, pp. 367–382.
- K. V. PALEM AND B. B. SIMONS (1993), *Scheduling time-critical instructions on RISC machines*, *ACM Trans. Program. Lang. Syst.*, 15, pp. 632–658.
- C. PHILLIPS, C. STEIN, AND J. WEIN (1998), *Minimizing average completion time in the presence of release dates*, *Math. Programming*, 82, pp. 199–223.
- M. QUEYRANNE (1993), *Structure of a simple scheduling polyhedron*, *Math. Programming*, 58, pp. 263–285.
- A. S. SCHULZ (1996a), *Polytopes and Scheduling*, Ph.D. thesis, Technische Universität Berlin, Berlin.
- A. S. SCHULZ (1996b), *Scheduling to minimize total weighted completion time: Performance guarantees of LP-based heuristics and lower bounds*, in *Integer Programming and Combinatorial Optimization*, Lecture Notes in Comput. Sci. 1084, W. H. Cunningham, S. T. McCormick, and M. Queyranne, eds., Springer, Berlin, pp. 301–315.
- A. S. SCHULZ AND M. SKUTELLA (1997), *Random-based scheduling: New approximations and LP lower bounds*, in *Randomization and Approximation Techniques in Computer Science*, Lecture Notes in Comput. Sci. 1269, J. Rolim, ed., Springer, Berlin, pp. 119–133.
- A. S. SCHULZ AND M. SKUTELLA (2002a), *The power of α -points in preemptive single machine scheduling*, *J. Sched.*, 5, pp. 121–133.
- A. S. SCHULZ AND M. SKUTELLA (2002b), *Scheduling unrelated machines by randomized rounding*, *SIAM J. Discrete Math.*, 15, pp. 450–469.
- P. SCHURMAN (1998), *A fully polynomial approximation scheme for a scheduling problem withintree-type precedence delays*, *Oper. Res. Lett.*, 23, pp. 9–11.
- E. D. WIKUM, D. C. LLEWELLYN, AND G. L. NEMHAUSER (1994), *One-machine generalized precedence constrained scheduling problems*, *Oper. Res. Lett.*, 16, pp. 87–99.
- L. A. WOLSEY (1985), *Mixed Integer Programming Formulations for Production Planning and Scheduling Problems*, invited talk at the 12th International Symposium on Mathematical Programming, Massachusetts Institute of Technology, Cambridge, MA.

OBLIVIOUS POLYNOMIAL EVALUATION*

MONI NAOR[†] AND BENNY PINKAS[‡]

Abstract. Oblivious polynomial evaluation is a protocol involving two parties, a sender whose input is a polynomial P , and a receiver whose input is a value α . At the end of the protocol the receiver learns $P(\alpha)$ and the sender learns nothing. We describe efficient constructions for this protocol, which are based on new intractability assumptions that are closely related to noisy polynomial reconstruction. Oblivious polynomial evaluation can be used as a primitive in many applications. We describe several such applications, including protocols for private comparison of data, for mutually authenticated key exchange based on (possibly weak) passwords, and for anonymous coupons.

Key words. cryptography, secure computation, noisy polynomial reconstruction

AMS subject classifications. 94A60, 11T71

DOI. 10.1137/S0097539704383633

1. Introduction. A secure computation protocol for a function $f(\cdot, \cdot)$ allows two parties, a receiver who knows x and a sender who knows y , to jointly compute the value of $f(x, y)$ in a way that does not reveal to each side more information than it can learn from $f(x, y)$. The fact that for every polynomially computable function $f(\cdot, \cdot)$ there exists such a (polynomially computable) protocol is one of the most remarkable achievements of research in foundations of cryptography. However, the resulting protocols are often not as efficient as one would desire, since the number of cryptographic operations that should be performed is proportional to the *size of the circuit computing $f(x, y)$* [60]. Even for relatively simple functions this may be prohibitively expensive. It is therefore interesting to investigate for which functions it is possible to come up with a protocol that does not emulate a circuit computing the function.

1.1. Oblivious polynomial evaluation. In the oblivious polynomial evaluation (OPE) problem the input of the sender is a polynomial P of degree k over some field \mathcal{F} . The receiver can learn the value $P(x)$ for any element $x \in \mathcal{F}$ without learning anything else about the polynomial P and without revealing to the sender any information about x (for the precise definition of “learning” and “information” see section 1.2). We find this problem to be a useful primitive. For example, as it can act as a cheap replacement for pseudorandom functions in case only k -wise independence is needed.

Strongly polynomial overhead The overhead of an algorithm is strongly polynomial if it is bounded by a polynomial function of the number of data items in the input,

*Received by the editors January 15, 2004; accepted for publication (in revised form) September 28, 2005; published electronically March 17, 2006. This paper is the full version of the sections that describe oblivious polynomial evaluation in “Oblivious Transfer and Polynomial Evaluation,” in Proceedings of the 31st ACM Symposium on the Theory of Computing, 1999.

<http://www.siam.org/journals/sicomp/35-5/38363.html>

[†]Department of Computer Science and Applied Mathematics, Weizmann Institute of Science, Rehovot 76100, Israel (moni.naor@weizmann.ac.il). Research supported by grant 356/94 from the Israel Science Foundation administered by the Israeli Academy of Sciences.

[‡]Department of Computer Science, University of Haifa, Haifa 31905, Israel (benny@cs.haifa.ac.il, benny@pinkas.net). Most of this work was done at the Weizmann Institute of Science and was supported by an Eshkol grant of the Israel Ministry of Science. Part of this work was done at HP Labs, Technion City, Haifa 32000, Israel.

rather than a function of the *size* of the input values (e.g., the number of bits of numerical input values). In combinatorial optimization it is common to look for strongly polynomial algorithms for different problems, for example, for linear programming.

The computational overhead of cryptographic protocols is usually measured as the number of *public key operations*. We use this term to denote cryptographic operations which we do not know how to implement based on the existence of one-way functions, and for which the best known constructions are based on trapdoor functions, or on similar primitives. (In the context of this paper we usually only measure the number of invocations of a 1-out-of-2 oblivious transfer protocol.) Counting only public key operations is justified since the overhead of public key operations depends on the length of their inputs, and is greater by orders of magnitude than the overhead of symmetric key operations (i.e., operations based on one-way functions). Furthermore, the separation result of Impagliazzo and Rudich [38], and subsequent work, hint that it is unlikely that these operations can be based on the existence of one-way functions.

We say that a cryptographic protocol is *strongly polynomial* if the following two properties hold: (1) the number of public key operations performed by the protocol is bounded by a polynomial function of a security parameter and of the number of inputs (but not their size), and (2) the length of the inputs to the public key operations is linear in the security parameter. Note that the number of *symmetric* key operations that the protocol performs can be polynomial in the size of its inputs.

Known methods of implementing oblivious polynomial evaluation include generic protocols for secure two-party computation (e.g., [60]), or using homomorphic encryption (e.g., using Paillier’s encryption system [56]). However, none of these methods is strongly polynomial in the sense defined above: the number of oblivious transfer operations in the generic construction, as well as the size of the input to the homomorphic encryption function, is linear in the size of the receiver’s input in the OPE protocol.

In contrast, the number of oblivious transfers (OT) used by the OPE protocols presented in this paper does not depend on the size of the underlying field: the length of the items transferred in the oblivious transfer protocols is of size $\log |\mathcal{F}|$, but they require only $O(1)$ public key operations per transfer. Namely, if $\log |\mathcal{F}|$ is longer than the length of the input of the OT protocol, then the items to be transferred in the OT protocol are encrypted using random keys, and the corresponding keys are transferred in the actual OT protocol.

An alternative formulation of the “strongly polynomial” requirement for cryptographic protocols is to allow only “black box” access to field operations, while hiding the underlying field from the parties. In addition, the number of invocations of the black box must be independent of the size of the inputs. Note that our construction satisfies this requirement, whereas generic constructions, or constructions based on homomorphic encryption, do not.

It is interesting to note that the work on randomizing polynomials [36] implies that any function which has an efficient branching program representation can be efficiently reduced to computing polynomials. We explain in section 4.3 how to reduce two-party secure computation of such functions to OPE. In particular, this implies strongly polynomial protocols for computing any function which has an efficient branching program representation.

Our protocols can also be readily applied to oblivious computation of $g^{P(x)}$, where P is a polynomial, with no increase in their overhead, as described in section 4.1. In comparison, binary circuits that evaluate this function must compute exponentiations and are very large, and the overhead of the resulting secure protocols is high.

(Protocols based on homomorphic encryption, however, can compute $g^{P(x)}$ without increasing their overhead.) Oblivious evaluation of $g^{P(x)}$ yields k -wise independent outputs that might be useful for distributed implementations of public key operations.

The noisy polynomial reconstruction problem. The protocols we describe are based on intractability assumptions related to the noisy polynomial reconstruction problem. While being related to other computational problems, this problem has not been used before as an intractability assumption in a cryptographic protocol. Section 2 describes in detail the assumptions and the background. Interestingly, following the publication of the conference version of our work [51] Kiayias and Yung designed different cryptographic primitives which are based on variants of the noisy polynomial reconstruction problem [40, 41].

Overhead independent of the degree of the polynomial. We describe a protocol (Protocol 3.4) whose computational overhead (as measured by the number of 1-out-of-2 oblivious transfers that are computed) is independent of the degree of the polynomial. That is, for any degree d , the number of oblivious transfers that are required for an oblivious evaluation of a polynomial of degree d is the same as for a linear polynomial. (Of course, the total computation overhead of the evaluation of the polynomial depends on the degree of the polynomial, but the number of public key operations is independent of the degree.)

Applications. We envision two types of applications for oblivious polynomial evaluation. One is whenever k -wise independence can replace full independence or pseudorandomness (i.e., oblivious evaluation of a pseudorandom function, as in [54]). Such property is required, for example, for the application of constructing anonymous coupons that enable anonymous usage of limited resources (e.g., for constructing an anonymous complaint box). The other type of application uses OPE for preserving anonymity when the receiver must compute the value of a polynomial at a certain point. Applications of this nature include a protocol that allows reliable and privacy preserving metering (described in section 4.4), a method for distributed generation of RSA keys, designed by Gilboa [26], and a protocol for private computation of the ID3 data mining algorithms [45] (in that case the polynomial is used for a Taylor approximation of the logarithm function).

1.2. Correctness and security definitions. We now get to the delicate business of defining the security of oblivious polynomial evaluation. The OPE functionality requires privacy for both receiver and sender. In an OPE protocol neither party learns anything more than is defined by the OPE functionality. The strongest way of formalizing this notion and ensuring simple composability of the protocols is through the definition of secure two-party computation (see, e.g., Goldreich [29] and papers on composability, e.g., [10, 11]). However, this definition is rather complex, while there are many applications that do not require the full power of the general definition and could use “non-ideal” protocols. We therefore prefer to use a relaxed definition for OPE, which ensures privacy for both parties but does not require the sender to commit to its input (i.e., to commit to the polynomial P). We call this definition *private computation*. The definition of private computation is relevant also to the case of malicious parties (and is therefore stronger than a definition for the semi-honest case only). It preserves the privacy of the clients, but does not require simulation of the joint distribution of the view of a malicious sender and the output of an honest receiver, as is required by the general definition of secure computation.

We claim that this relaxation is justified by efficiency considerations, in particular when constructing specific OPE protocols rather than black-box reductions of OPE

to other primitives. Furthermore, the definition of private computation is standard for related primitives such as oblivious transfer [58, 21, 53] or private information retrieval (PIR) [13, 43] (note that we present a reduction of OPE to oblivious transfer). Note also that the definition of private computation is equivalent to the definition of secure computation in the case of semi-honest parties. Furthermore, we deal with a receiver-sender (aka. client-server) scenario, where only one party, the receiver, has an output in the protocol. Therefore, the two definitions are equivalent with respect to a malicious *client*, as there is no issue of simulating the joint distribution of the client's view and the server's output.

The requirements of a private OPE protocol can be divided into *correctness*, *receiver privacy*, and *server privacy*. Let us first define these properties independently and then define a private OPE protocol as a protocol satisfying these three properties. In the definitions, the running time of polynomial time algorithms is polynomial in the size of their inputs, as well as in $\log |\mathcal{F}|$, where \mathcal{F} is the field in which the polynomial P is defined, and in a security parameter k . (Note that the length of representations of elements in \mathcal{F} must be polynomial in the security parameter since otherwise the cryptographic operations might be insecure given adversaries with poly- $\log |\mathcal{F}|$ running time.) We don't require in the definitions themselves that the number of public key operations is independent of \mathcal{F} . To simplify the notation we also omit any reference to auxiliary inputs.

We first define the input and output for the OPE functionality as a two party protocol run between a receiver and a sender over a field \mathcal{F} .

- Input:
 - Receiver: an input $x \in \mathcal{F}$.
 - Sender: a polynomial P defined over \mathcal{F} .
- Output:
 - Receiver: $P(x)$.
 - Sender: nothing.

DEFINITION 1.1 (correctness, or functionality). *At the end of the protocol the receiver obtains the output of the OPE functionality, namely, $P(x)$.*

The definition of the receiver's privacy is simplified by the fact that the sender gets no output. It is as follows.

DEFINITION 1.2 (receiver's privacy—indistinguishability). *For any probabilistic polynomial time \mathcal{B}' executing the sender's part, for any x and x' in \mathcal{F} , the views that \mathcal{B}' sees in case the receiver's input is x and in case the receiver's input is x' are computationally indistinguishable.*

The definition of sender's privacy is a bit trickier, since the receiver obtains some information, and we want to say that the receiver does not get more or different information than $P(x)$. We compare the protocol to the *ideal implementation*. In the ideal implementation there is a trusted third party Charlie, which gets the sender's polynomial P and the receiver's request x and gives $P(x)$ to the receiver. The privacy requirement is that the protocol does not leak to the receiver more information than in the ideal implementation.

DEFINITION 1.3 (sender's security—comparison with the ideal model). *For any probabilistic polynomial-time machine \mathcal{A}' substituting the receiver, there exists a probabilistic polynomial-time machine \mathcal{A}'' that plays the receiver's role in the ideal implementation, such that the view of \mathcal{A}' and the output of \mathcal{A}'' are computationally indistinguishable.*

DEFINITION 1.4 (private OPE protocol). *A two-party protocol satisfying Definitions 1.1, 1.2 and 1.3.*

Note that the definition of receiver privacy does *not* preclude the sender from cheating by using a polynomial of degree higher than the degree of P (and therefore it might not be possible to extract from the sender a degree k polynomial). We do not require that the sender be committed to a single polynomial, or that the receiver could verify that the value she receives corresponds to such a polynomial. Our construction allows such cheating; however, in many applications (including the ones described in this paper) this is immaterial.

As a side note we observe that a possible approach for ensuring correctness could use the verifiable secret sharing (VSS) schemes of Feldman and of Pedersen [23, 57], which let the sender commit to a single polynomial P before engaging in the OPE protocol. Since two polynomials of degree d agree in at most d locations, an OPE invocation in which the sender lets the user evaluate a different polynomial P' of the same degree is revealed with probability $1 - d/|\mathcal{F}|$ by a receiver that evaluates the polynomial at a *random* point. This approach does not work, however, if the sender has some information about the distribution of points in which the user might compute P .

Shared output OPE. A variant of OPE which might be useful in many applications (e.g., [26, 45]) is one where the output of the polynomial is shared between the two parties. As in the basic OPE protocol the input of the sender is a polynomial $P()$ and the input of the receiver is x . However, unlike the basic OPE protocol both parties have random outputs which sum up to $P(x)$. In other words, the sender and receiver have output y_S and y_R , respectively, such that each of these values is uniformly distributed in \mathcal{F} and it also holds that $y_S + y_R = P(x)$.

There is a simple reduction from shared output OPE to basic OPE. Namely, given an input polynomial $P()$, the sender chooses a random $y_S \in \mathcal{F}$ and defines $P'(x) = P(x) - y_S$. The parties then perform an OPE of $P'(x)$ and the receiver sets $y_R = P'(x)$.

1.3. Related work.

1.3.1. Oblivious transfer. The basic cryptographic primitive that is used by the protocols for oblivious polynomial evaluation is *oblivious transfer*. The notion of 1-out-of-2 oblivious transfer was suggested by Even, Goldreich and Lempel [21] as a generalization of Rabin's oblivious transfer [58]. A protocol for 1-out-of-2 OT involves a sender, which has two inputs x_0 and x_1 , and a receiver whose input is a single bit, $b \in \{0, 1\}$. At the end of the protocol the receiver learns x_b and nothing about x_{1-b} , while the sender learns nothing about b . More generally, a k -out-of- N OT functionality is defined in the following way.

DEFINITION 1.5. (k -out-of- N oblivious transfer functionality).

- *Parameters:* Integers $N > k \geq 1$.
- *Input:*
 - *Sender:* N inputs x_1, \dots, x_N .
 - *Receiver:* k inputs i_1, \dots, i_k which are integer values in the range $[1, N]$.
- *Output:*
 - *Sender:* nothing.
 - *Receiver:* x_{i_1}, \dots, x_{i_k} .

For a discussion of OT see Goldreich [29]. 1-out-of- N oblivious transfer was introduced by Brassard, Crépeau and Robert [8, 9] under the name ANDOS (all or nothing disclosure of secrets). They used information theoretic reductions to construct 1-out-of- N protocols from $N - 1$ invocations of a 1-out-of-2 protocol (it was later shown that such reductions must use at least $\Omega(N)$ invocations of 1-out-of-2 OT in order

to preserve the information theoretic security [17]). Goldreich and Vainish [33] and Kilian [42] showed that oblivious transfer enables general secure two-party computation, with no additional assumptions (with security against semi-honest and malicious adversaries, respectively).

This paper describes reductions of oblivious polynomial evaluation to an ideal implementation of the OT functionality of Definition 1.5. Based on composition theorems (see, e.g., [29, 10, 11]) the ideal functionality can be replaced with actual implementations of OT. Such constructions of OT can be based on physical assumptions, such as the use of a noisy channel (see, e.g., [15]), or on computational assumptions. Constructions of OT based on computational assumptions can be divided to different categories, as listed below, according to the security definitions that they satisfy. The OPE constructions can be based on OT protocols from any of these categories. (Of course, security depends on the security of the oblivious transfer protocol. In particular, all OT protocols we describe provide security against semi-honest adversaries. All but the first category provide *privacy* in the face of malicious parties.)

(i) OT protocols which provide security in the semi-honest model. These include basic protocols based on the EGL (Even–Goldreich–Lempel) paradigm [21], which are based on using a public key encryption system with the additional property that the distribution of ciphertexts is independent of the encryption key (the original EGL protocol uses trapdoor permutations with a special property, but its structure is similar). In these protocols the receiver sends two encryption keys PK_0 and PK_1 , while knowing only a single decryption key, corresponding to PK_b . The sender encrypts x_0 using the key PK_0 , and encrypts x_1 using the key PK_1 . The receiver then decrypts x_b but cannot decrypt x_{1-b} . Protocols based on this paradigm include the construction suggested by Bellare and Micali [4], and generic constructions based on the existence of trapdoor permutations.

OT protocols of this type can be made secure with respect to malicious parties by applying the GMW (Goldreich–Micali–Wigderson) [31] paradigm, i.e., “compiling” a semi-honest protocol using generic techniques to obtain a protocol which ensures that the operation of the parties follows the protocol. This is usually done by adding zero-knowledge proofs in which the parties prove that they operate according to the protocol. If the zero-knowledge proofs enable extraction then the protocols are simulatable, but only for a single invocation at a time, rather than for parallel or concurrent invocations of the protocol.

(ii) OT protocols which provide information-theoretic security for the sender with respect to a corrupt receiver, and computational security for the receiver (e.g., the two round protocols of [52, 1, 39]). Although these protocols provide information-theoretic security, they do not enable extraction of the receiver’s input. Therefore they do not enable easy simulation of the output that is obtained by the receiver. (The GMW paradigm can be applied to this type of protocol as well, but usually with considerable degradation in efficiency. In addition, the information-theoretic security might be lost.)

(iii) Fully simulatable OT protocols, in particular for parallel or concurrent invocations. These include the concurrent OT protocol of [24] (which assumes that the inputs are independent), the universally composable protocols of [11], and the universally composable committed OT of [25].

(iv) Protocols based on the random oracle model. In this model it is possible to design very efficient protocols based on the EGL or the Bellare–Micali paradigms. These protocols are secure against malicious parties and fully simulatable. Their security relies, however, on the random oracle model.

Most constructions of OT protocols that are described in the literature are of 1-out-of-2 OT. Efficient implementations of 1-out-of- N and k -out-of- N oblivious transfer protocols are described in [53]. These protocols are based on efficient computationally secure reductions to 1-out-of-2 oblivious transfer. A 1-out-of- N oblivious transfer is reduced to $\log N$ invocations of 1-out-of-2 OT, and the k -out-of- N protocol is considerably more efficient than k repetitions of 1-out-of- N oblivious transfer. Furthermore, more recent constructions [52] reduce the amortized overhead of oblivious transfer, if multiple invocations of this protocol should be run (as is the case in the OPE constructions). A direct implementation of 1-out-of- N OT, e.g., by the protocols which provide information-theoretic security for the sender [52, 1, 39], is quite efficient for the receiver which has to do only $O(1)$ work, while the sender has to perform $O(N)$ exponentiations.

It was recently shown how to extend oblivious transfer in the sense that two parties can execute a large number of OTs (a number polynomial in k , where k is a security parameter of a pseudorandom function), at the cost of running only k OTs and executing additional invocations of symmetric cryptographic functions for every OT [35] (this is an efficient realization of a generic construction of Beaver [2]). The security of this construction is based on a nonstandard assumption (or alternatively on the random oracle assumption), and security against malicious parties is obtained using a cut-and-choose method that involves running multiple invocations of the system.

In our work we measure the computational overhead by the number of OTs that are executed by the parties. The use of this criterion makes sense since all other operations are either arithmetic or are symmetric crypto operations, which are considerably more efficient. It is preferable to minimize the number of OT operations even given the recent work on extending oblivious transfer, since that construction depends on a new assumption, involves some additional constants, and requires a cut-and-choose solution against malicious parties.

We present a protocol that uses a minimal number of OTs in the sense that this number is independent of the size of the field and of the degree of the polynomial. We assume that each 1-out-of-2 OT operation is atomic and can accommodate an input of arbitrary size (this is justified since the OT can be used to transfer one of two keys whose length is equal to the security parameter, and these keys can be used to encrypt each of the two inputs, which can be of arbitrary size).

1.3.2. Secure two-party computation. The idea of secure two-party computation was introduced by Yao [60]. His construction enables one party, the sender, to define a function F and enable another party, the receiver, to compute the value of F at a single point x without learning anything else about F , and without disclosing to the sender any information about x . The construction is based on describing F as a binary circuit and evaluating the circuit. Its computational overhead is composed of running an oblivious transfer protocol for every input wire of the circuit, and computing a pseudorandom function for every gate. The bulk of the communication overhead is incurred by sending a table, of size linear in the security parameter of the pseudorandom function, for every gate. The overhead, therefore, strongly depends on the size of the representation of F as a binary circuit. In the case of a polynomial of degree d this circuit should compute x^d and its size is $O(|x|^2 \cdot \log d)$. In particular, the size of the circuit depends on the size of the field over which the polynomial is defined. (It is also possible to construct a circuit that uses an FFT approach for computing x^d . The size of this circuit, too, depends on the size of the field over which the polynomial is defined.)

In another generic construction, Naor and Nissim [49] show that any two-party protocol can be transformed into a secure protocol with the effect that a protocol with communication complexity of c bits is transformed to a secure protocol which performs c invocations of oblivious transfer (or SPIR) from a database of length 2^c . A simple (insecure) protocol for oblivious polynomial evaluation has a single round and a communication overhead of $\log |\mathcal{F}|$ bits (the receiver simply sends x to the sender). Applying the Naor–Nissim transformation to this protocol results in a secure protocol that executes an OT/SPIR out of a table of $|\mathcal{F}|$ elements. Namely, the server constructs a table of $|\mathcal{F}|$ items, containing the value of $P(x)$ for every $x \in \mathcal{F}$. The receiver then reads a single entry of this table using OT or SPIR. This protocol is definitely not strongly polynomial as its overhead is linear in $|\mathcal{F}|$.

2. Intractability assumptions. This section contains definitions of two new pseudorandomness assumptions. They are later used for constructing protocols for oblivious polynomial evaluation. The assumptions are closely related to the noisy polynomial reconstruction problem, or the list decoding problem of Reed–Solomon codes. We first describe this well-known problem, and then introduce the pseudorandomness assumptions.

2.1. The noisy polynomial reconstruction problem. The noisy polynomial reconstruction problem is described by the following definition.

DEFINITION 2.1 (polynomial reconstruction). *A polynomial reconstruction algorithm has the following functionality:*

- *Input: Integers k and t , and n points $\{(x_i, y_i)\}_{i=1}^n$, where $x_i, y_i \in \mathcal{F}$ (\mathcal{F} is a field).*
- *Output: Any univariate polynomial P of degree at most k such that $P(x_i) = y_i$ for at least t values $i \in [1, n]$.*

The noisy polynomial reconstruction problem is related to the list decoding problem, which is motivated by coding theory and was first defined by Elias [20] (and sometimes also termed as the bounded-distance decoding problem). The input to this problem is a received word, and the output is a list of all code words that are within some distance from the received word. For the case of Reed–Solomon codes the list decoding problem can be formulated as follows.

DEFINITION 2.2 (polynomial list reconstruction). *A polynomial list reconstruction algorithm has the following functionality.*

- *Input: Integers k and t , and n points $\{(x_i, y_i)\}_{i=1}^n$, where $x_i, y_i \in \mathcal{F}$ (\mathcal{F} is a field).*
- *Output: All univariate polynomials P of degree at most k such that $P(x_i) = y_i$ for at least t values $i \in [1, n]$.*

For given values of k and n , and in particular for a given message rate k/n , it is preferable to obtain solutions for minimal values of t . The classic algorithm of Berlekamp and Massey (see, e.g., [48]) solves the polynomial reconstruction problem in polynomial time for $t \geq \frac{n+k}{2}$ (in this range there is a unique solution). Sudan [59] presented a polynomial algorithm that works if $t \geq \sqrt{2kn}$, and later Guruswami and Sudan [34] presented a polynomial algorithm that solves the problem for $t \geq \sqrt{kn}$, and thus improves upon the Berlekamp–Massey algorithm for every value of the message rate k/n . (Later, Coppersmith and Sudan [14] showed an improved noisy interpolation algorithm which removes random errors applied to curves of the form $\langle x, p_1(x), p_2(x), \dots, p_c(x) \rangle$, where p_1, \dots, p_c are polynomials. We are interested in the case $c = 1$, for which this algorithm is not better than the Guruswami–Sudan algorithm.)

The “polynomial list reconstruction” problem is defined as a worst case problem regarding the *number* of polynomials that might appear in the solution list. In other words, the definition requires that all polynomials within a given distance be listed. Goldreich et al. [32] have shown that for $t > \sqrt{kn}$ the number of possible polynomials in the solution to the problem is bounded by a polynomial in n . We are more interested in the problem of finding just a single polynomial that fits t or more of the given n points, since our constructions use random instances for which it holds with high probability that the corresponding noisy polynomial reconstruction problem has a single solution.

We note that given an input to a noisy polynomial reconstruction problem it is possible to randomize it to obtain an input that corresponds to a random polynomial. Specifically, for parameters k and t and given n points $\{(x_i, y_i)\}_{i=1}^n$ the randomization is achieved by choosing a random polynomial R of degree k and constructing a new instance of the problem with input $\{(x_i, y_i + R(x_i))\}_{i=1}^n$. While this is by no means a reduction from the worst case problem to the average case (since it only randomizes the polynomial but not the noise), it might hint that solving the problem in the average case might not be much simpler than solving it in the worst case.

2.2. The intractability assumptions—pseudorandomness. We present two new intractability assumptions. The first assumption is equivalent to a conjecture that given a randomly chosen input to the polynomial list reconstruction problem the value of the polynomial at $x = 0$ is pseudorandom. The second assumption states that the value $P(0)$ is pseudorandom even given some additional hints about the location of the values of P . Section 3 describes OPE protocols that are based on the assumptions.

The intractability assumptions depend on the following parameters:

- (i) \mathcal{F} , the field over which the polynomial is defined.
- (ii) k , the degree of the hidden polynomial.
- (iii) n , the number of correct values of the polynomial, which is also the number of queries made in the oblivious evaluation protocol. (This parameter corresponds to t in the definition of the polynomial list reconstruction problem, Definition 2.2. We change the notation to agree with the notation used later in this paper.)
- (iv) m , the expansion ratio (namely, the ratio between the total number of points and n). This parameter corresponds to n/t in Definition 2.2.

Setting precise parameter sizes to satisfy the intractability assumptions is beyond the scope of this paper, and we therefore do not make any precise recommendations with regard to parameter sizes.

The first intractability assumption. This intractability assumption simply assumes that given an input to the polynomial list reconstruction problem, with all x_i being distinct, the value of the polynomial at $x = 0$ is pseudorandom. That is, it is infeasible to distinguish between two such inputs corresponding to polynomials with different values at $x = 0$. To define this more formally, we use the following notation:

Let $A_{n,m}^{k,\alpha}$ denote the probability distribution of sets generated in the following way:

1. Pick a random polynomial P over \mathcal{F} , of degree at most k , for which it holds that $P(0) = \alpha$.
2. Generate nm random values x_1, \dots, x_{nm} in \mathcal{F} subject to the constraint that all x_i values are distinct and different from 0.
3. Choose a random subset S of n different indices in $[1, nm]$, and set $y_i = P(x_i)$ for all $i \in S$. For every $i \notin S$ set y_i to be a random value in \mathcal{F} .
4. Output the set $\{(x_i, y_i)\}_{i=1}^{nm}$.

The pseudorandomness assumption is based on the notion of computationally indistinguishability, as defined by Goldreich in [28, page 104]. It sets the size of the parameters to be polynomial in a security parameter ℓ , and requires that the resulting probability ensembles are computationally indistinguishable.

Assumption 1 (first pseudorandomness assumption). Let ℓ be a security parameter, and let $n(\ell), m(\ell), k(\ell), F(\ell)$ be polynomially bounded functions that define the parameters n, m, k and the description of the field \mathcal{F} . Let $\alpha(\ell)$ and $\alpha'(\ell)$ be any polynomially bounded functions defining elements of the field \mathcal{F} , and define $\alpha = \alpha(\ell)$ and $\alpha' = \alpha'(\ell)$. Let $\mathcal{A}_{n,m}^{k,\alpha}$ and $\mathcal{A}_{n,m}^{k,\alpha'}$ be random variables that are chosen according to the distributions $A_{n,m}^{k,\alpha}$ and $A_{n,m}^{k,\alpha'}$, respectively.

The assumption is that the probability ensembles $\{\mathcal{A}_{n,m}^{k,\alpha}\}$ and $\{\mathcal{A}_{n,m}^{k,\alpha'}\}$ are computationally indistinguishable for adversaries whose running time is polynomial in the security parameter ℓ .

Pseudorandomness Assumption 1 is related to the assumption that polynomial list reconstruction is hard. Assumption 1 is stronger, since in addition to assuming that reconstructing the polynomial is hard, it assumes that all x_i are distinct and that it is even hard to learn information about the value of the polynomial at 0.

Assumption 1 is weaker than an assumption that states that it is hard to distinguish between any probability ensemble $\{\mathcal{A}_{n,m}^{k,\alpha}\}$ and a probability ensemble that generates sets with the same number of *random* (x, y) values. If the latter assumption is true, then so is Assumption 1. (Assume that Assumption 1 does not hold. Then there is a distinguisher such that the probability of its output being 1 given an input from $\{\mathcal{A}_{n,m}^{k,\alpha}\}$ has a nonnegligible difference from the probability of it having a 1 output given an input from $\{\mathcal{A}_{n,m}^{k,\alpha'}\}$. The probability of a 1 output given an input from the “random” ensemble must have a nonnegligible difference from at least one of these two probabilities.) The converse might not be true. It might be that it is easy to distinguish between an input from the “random” ensemble and an input from $\{\mathcal{A}_{n,m}^{k,\alpha}\}$, yet for all α values, the inputs from the different $\{\mathcal{A}_{n,m}^{k,\alpha}\}$ ensembles are indistinguishable.

The pseudorandomness assumption is of course false for any choice of parameters for which the polynomial list reconstruction problem is easy, e.g., when $m < n/k$. (This corresponds to the number of correct points, n , being more than the square root of the total number of points, nm , times the degree of polynomial, k . This equation therefore agrees with the threshold for which the noisy polynomial problem is easy.) In the other direction, it is an open problem to find a reduction from the polynomial list reconstruction to the assumption.

The second intractability assumption. The second assumption states that given n sets with m points in every set, such that a polynomial P agrees with at least one point in each set, the value of $P(0)$ is pseudorandom. Namely, the total number of points, and the number of correct points, are as in pseudorandomness Assumption 1, but in addition there is a partition into sets and it is promised that the polynomial P agrees with at least one point in each set. It is hard to come up with a reduction to this assumption from pseudorandomness Assumption 1 since the reduction must map each of the n points of P into a different set. The new assumption seems stronger than pseudorandomness Assumption 1 since the problem is easier. Namely, the adversary is given an additional hint—the partition into sets containing at least one correct value of the polynomial.

The definition of the assumption uses the following notation.

Let $C_{n,m}^{k,\alpha}$ denote the probability distribution of sets generated in the following way:

1. Pick a random polynomial P over \mathcal{F} , of degree at most k , for which it holds that $P(0) = \alpha$.
2. Generate nm random values x_1, \dots, x_{nm} in \mathcal{F} subject to the constraint that all x_i values are distinct and different from 0.
3. Choose a random subset S of n different indices in $[1, nm]$, and set $y_i = P(x_i)$ for all $i \in S$. For every $i \notin S$ set y_i to be a random value in \mathcal{F} .
4. Partition the nm (x_i, y_i) pairs to n random subsets subject to the following constraints:
 - (i) The subsets are disjoint.
 - (ii) Each subset contains exactly one pair whose index is in S (and therefore for this pair it holds that $y_i = P(x_i)$).
 - (iii) Each subset contains exactly $m-1$ pairs whose indices are not in S (and therefore have a random y_i value).

Output the resulting subsets.

The intractability assumption says that for any α, α' the two probability ensembles $\{C_{n,m}^{k,\alpha}\}, \{C_{n,m}^{k,\alpha'}\}$ are computationally indistinguishable. It depends on the parameters F, k, m , and n .

Assumption 2 (second pseudo-randomness assumption). Let ℓ be a security parameter, and let $n(\ell), m(\ell), k(\ell), F(\ell)$ be polynomially bounded functions that define the parameters n, m, k and the description of the field \mathcal{F} . Let $\alpha(\ell)$ and $\alpha'(\ell)$ be any polynomially bounded functions defining elements of the field \mathcal{F} , and define $\alpha = \alpha(\ell)$ and $\alpha' = \alpha'(\ell)$. Let $C_{n,m}^{k,\alpha}$ and $C_{n,m}^{k,\alpha'}$ be random variables that are chosen according to the distributions $C_{n,m}^{k,\alpha}$ and $C_{n,m}^{k,\alpha'}$, respectively.

The assumption is that the probability ensembles $\{C_{n,m}^{k,\alpha}\}$ and $\{C_{n,m}^{k,\alpha'}\}$ are computationally indistinguishable.

As with Assumption 1 there is an easy reduction from the problem of breaking this pseudorandomness assumption to the noisy polynomial reconstruction problem. In addition, if pseudorandomness Assumption 1 does not hold in the worst case, then pseudorandomness Assumption 2 does not hold in the worst case (since any input for the latter can be transformed to an input for the first assumption by simply ignoring the partition into subsets). This reduction is also true in the average case: consider all sets generated by the distributions $A_{n,m}^{k,\alpha}$ and $C_{n,m}^{k,\alpha}$ subject to the constraint that no more than n points in a set agree with the polynomial P (the probability that this property does not hold for a specific set is at most $n(m-1)/|\mathcal{F}|$, which we consider to be negligible). Denote the resulting collections of sets as $\hat{A}_{n,m}^{k,\alpha}$ and $\hat{C}_{n,m}^{k,\alpha}$. A set in $\hat{A}_{n,m}^{k,\alpha}$ is matched to a set in $\hat{C}_{n,m}^{k,\alpha}$ if they contain exactly the same points. Each of the sets in $\hat{A}_{n,m}^{k,\alpha}$ is therefore matched with the exactly same number of sets in $\hat{C}_{n,m}^{k,\alpha}$, and no two sets in $\hat{A}_{n,m}^{k,\alpha}$ are matched with the same set in $\hat{C}_{n,m}^{k,\alpha}$. Now, picking a random set in $\hat{C}_{n,m}^{k,\alpha}$ and removing the partition into subsets results in the matching set in $\hat{A}_{n,m}^{k,\alpha}$. Therefore, this procedure has the same probability of hitting any set in $\hat{A}_{n,m}^{k,\alpha}$.

It is an interesting open problem to provide a reduction to pseudorandomness Assumption 2 from pseudorandomness Assumption 1, or from the polynomial reconstruction problem. The problem with designing such a reduction is that its output should comply with the input distribution of pseudorandomness Assumption 2, namely, in each subset there should be (with high probability) a single value of the polynomial P .

Remark. The pseudorandomness assumption that was used in an earlier version of this work [51] was broken by Bleichenbacher and Nguyen [6] and by Boneh [7]. The assumption was similar to pseudorandomness Assumption 2, but with the additional requirement that in each subset of the sets generated by $C_{n,m}^{k,\alpha}$ all the points have the *same x coordinate*. This property enables reduction of this problem to an instance of the lattice shortest vector problem, and solving it using the LLL algorithm [44]. We remark that it is unknown how to employ this attack against pseudorandomness Assumption 2. Furthermore, the overhead of the oblivious evaluation scheme that is based on pseudorandomness Assumption 2 is not greater than that of the scheme that is based on the broken assumption.

3. Protocols for oblivious polynomial evaluation. This section describes protocols for oblivious evaluation of polynomials. The protocols reduce computing the OPE functionality to computing an OT functionality. We first describe a generic OPE protocol (Protocol 3.1), and then describe two instantiations of the generic protocol using each of the two pseudorandomness assumptions (Protocols 3.2 and 3.3, respectively). These protocols provide privacy against semi-honest or malicious senders, and against semi-honest receivers.

The protocols ensure that a malicious receiver learns at most a single linear equation of the coefficients of the polynomial, but this equation might not correspond to a legitimate value of the polynomial. We show that if the polynomial P that is evaluated is linear, then the protocols are secure even against malicious receivers, since the linear equation always corresponds to a value of the polynomial.

Security against malicious receivers, and improved efficiency. We then describe Protocol 3.4 which is secure against malicious receivers. Furthermore, Protocol 3.4, although a little more complicated conceptually, has computational overhead which is better than that of the previous protocols: the number of oblivious transfers is independent of the degree of the polynomial P and is equal to the number of oblivious transfers that are required in the case of a linear polynomial.

3.1. A generic protocol. All the protocols involve a receiver A and a sender B and have the following specifications:

- Input:
 - Sender: a polynomial $P(y) = \sum_{i=0}^{d_P} b_i y^i$ of degree d_P in the field \mathcal{F} .
 - Receiver: a value $\alpha \in \mathcal{F}$.
- Output:
 - Sender: nothing.
 - Receiver: $P(\alpha)$.
- Protocol security parameters: m, k , which are discussed below.

At the end of the protocol the parties learn nothing but their specified outputs. The generic protocol (Protocol 3.1) is described in Figure 1. It is based on the sender hiding P in a *bivariate* polynomial, and running oblivious transfer protocols with the receiver to reveal to her just enough information to enable the computation of $P(\alpha)$. (In this respect, the protocol is somewhat similar to the instance hiding construction of [3, 46].)

Note that the protocol uses a polynomial $Q(x, y)$ which is defined by $d + d_P + 1$ coefficients only, whereas a random bivariate polynomial of the same degrees is defined by $(d + 1)(d_P + 1)$ coefficients.

The only step that has to be further specified is step 3, in which the receiver learns $d_R + 1$ values of R . This is the only step that involves interaction between the two parties, and as such determines the overhead of the protocol. Below are descriptions

PROTOCOL 3.1 (a generic protocol for oblivious polynomial evaluation).

- The sender hides P in a bivariate polynomial:** (see Figure 2) The sender generates a random masking polynomial $P_x(x)$ of degree d , s.t. $P_x(0) = 0$. Namely, $P_x(x) = \sum_{i=1}^d a_i x^i$. The parameter d equals the degree of P multiplied by the security parameter k (i.e., $d = k \cdot d_P$).

The sender then defines a bivariate polynomial

$$Q(x, y) = P_x(x) + P(y) = \sum_{i=1}^d a_i x^i + \sum_{i=0}^{d_P} b_i y^i$$

for which it holds that $\forall y \ Q(0, y) = P(y)$.

- The receiver hides α in a univariate polynomial:** (see Figure 3) The receiver chooses a random polynomial S of degree k , such that $S(0) = \alpha$. The receiver's plan is to use the univariate polynomial $R(x) = Q(x, S(x))$ in order to learn $P(\alpha)$: it holds that $R(0) = Q(0, S(0)) = P(S(0)) = P(\alpha)$ and, therefore, if the receiver is able to interpolate R she can learn $R(0) = P(\alpha)$. The degree of R is $d_R = d = k \cdot d_P$.
- The receiver learns points of R :** The receiver learns $d_R + 1$ values of the form $(x_i, R(x_i))$.
- The receiver computes $P(\alpha)$:** The receiver uses the values of R that it learned to interpolate $R(0) = P(\alpha)$.

FIG. 1. A generic protocol for oblivious polynomial evaluation.

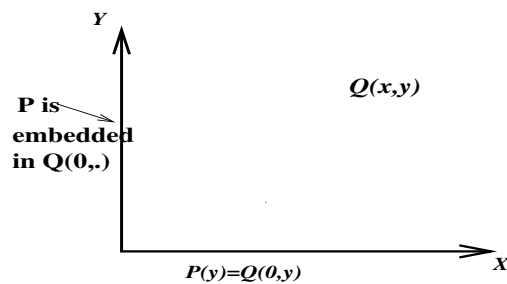


FIG. 2. The polynomial P embedded in the bivariate polynomial Q s.t. $Q(0, y) = P(y)$.

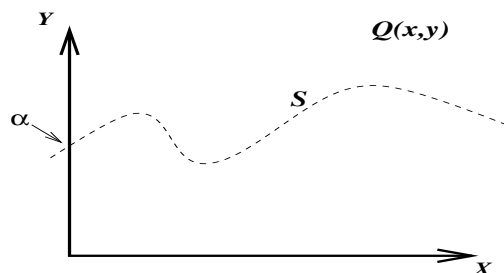


FIG. 3. The polynomial S defines a polynomial R s.t. $R(0) = Q(0, S(0)) = Q(0, \alpha) = P(\alpha)$.

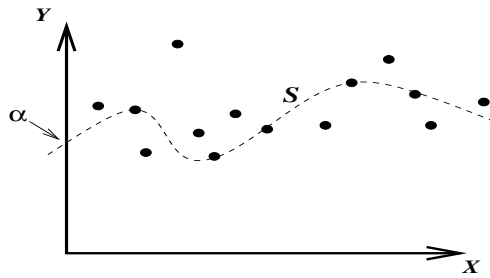


FIG. 4. The receiver uses n -out-of- N OT to learn n values of R hidden between N values.

of two protocols for oblivious polynomial computation in which step 3 is based on each of the two pseudorandomness assumptions.

3.2. Detailed protocols. This section describes instantiations of Protocol 3.1 based on the two pseudorandomness assumptions.

3.2.1. A protocol based on pseudorandomness Assumption 1. The first instantiation is a reduction to an n -out-of- N oblivious transfer protocol (see, e.g., [53]), in which the receiver learns n of the N inputs of the sender while hiding these values from the sender.

PROTOCOL 3.2 (oblivious polynomial evaluation based on Assumption 1). *The protocol is the generic protocol (Protocol 3.1), where the third step is run as follows:*

1. The receiver sets $n = d_R + 1 = d + 1 = kd_P + 1$ and chooses $N = nm$ distinct random values $x_1, \dots, x_N \in \mathcal{F}$, all different from 0.
2. The receiver chooses a random set T of n distinct indices $1 \leq i_1, \dots, i_n \leq N$. She then defines N values y_i , for $1 \leq i \leq N$. The value y_i is defined as $S(x_i)$ if i is in T , and is a random value in \mathcal{F} otherwise.
3. The receiver sends the N points $\{(x_i, y_i)\}_{i=1}^N$ to the sender.
4. The receiver and sender execute an n -out-of- N oblivious transfer protocol for the N values $Q(x_1, y_1), \dots, Q(x_N, y_N)$ (see Figure 4). The receiver chooses to learn $\{Q(x_i, y_i)\}_{i \in T}$.

The correctness of the protocol is based on observing that the receiver learns $d_R + 1$ values of the polynomial R , and can, therefore, interpolate R and compute $R(0) = P(\alpha)$. We prove the security of the protocol in section 3.3 below.

3.2.2. A protocol based on pseudorandomness Assumption 2. The second protocol is based on using n sets of points, such that each set contains a point of the polynomial R . The receiver runs an independent oblivious transfer protocol for each set, in which it learns a value of R .

PROTOCOL 3.3 (oblivious polynomial evaluation based on Assumption 2). *The protocol is the generic protocol (Protocol 3.1), where the third step is run as follows:*

1. The receiver sets $n = d_R + 1 = kd_P + 1$, defines $N = nm$ and chooses N distinct random values $x_1, \dots, x_N \in \mathcal{F}$, all different from 0.
2. The receiver chooses a set T of n randomly-chosen distinct indices $1 \leq i_1, i_2, \dots, i_n \leq N$, subject to the constraint that $(j - 1)m + 1 \leq i_j \leq jm$ for $1 \leq j \leq n$.
3. The receiver defines N values y_i , for $1 \leq i \leq N$. The value y_i is defined as $S(x_i)$ if i is in T , and is random otherwise.
4. The receiver partitions the N pairs $\{(x_i, y_i)\}_{i=1}^N$ into n subsets B_1, \dots, B_n , where subset B_j contains the m pairs indexed $(j - 1)m + 1$ to jm . This means that each subset B_j contains exactly one pair from T .

5. The receiver sends the n subsets B_1, \dots, B_n to the sender.
6. The receiver and sender execute n protocols of 1-out-of- m oblivious transfer, one for each subset. The protocol for subset B_j enables the receiver to learn one of the values $Q(x_i, y_i)$ for the m points (x_i, y_i) in B_j . (The receiver should choose to learn $Q(x_i, y_i)$ for which $y_i = P(x_i)$.)

3.2.3. Properties of the protocols.

Correctness. It is straightforward to verify that both protocols enable the receiver to obtain any value $P(\alpha)$ she desires. She can do this by choosing a polynomial S for which $S(0) = \alpha$, learning $d_R + 1$ values of R , and interpolating $R(0) = P(\alpha)$.

Complexity. The main overhead of the protocols, in terms of both computation and communication, is the overhead of the oblivious transfer stage. The overhead depends on the degree of P and on the security parameters k and m . Namely, the overhead of each protocol is that of running the following primitives:

- (i) Protocol 3.2 (based on Assumption 1): running a single invocation of $(kd_P + 1)$ -out-of- $[(kd_P + 1) \cdot m]$ oblivious transfer.
- (ii) Protocol 3.3 (based on Assumption 2): running $(kd_P + 1)$ invocations of 1-out-of- m oblivious transfer.

The actual overhead of the protocol depends on the value that is set to the parameter m , as a function of k and d_P , in order for the relevant security assumption to hold (this value might be different in each of the protocols). It might seem that Protocol 3.3, which uses $(kd_P + 1)$ invocations of 1-out-of- m oblivious transfer, is always inferior to Protocol 3.2, which uses a single invocation of $(kd_P + 1)$ -out-of- $[(kd_P + 1) \cdot m]$ oblivious transfer. Also, the fact that in Protocol 3.3 each subset is known to contain a value of the polynomial S might make the task of attacking the receiver easier, and require the use of larger parameter m to ensure the security of the protocol. We describe both protocols since the choice of parameters k and m might be different in the two cases and result in Protocol 3.3 being the more efficient. Also, recent work on extending oblivious transfer [35] leads to more efficient implementation of multiple invocations of OT.

We note that the multiple invocations of the oblivious transfer protocol in Protocol 3.3 can be run in parallel. The protocol is secure if the specific oblivious transfer protocol which is used can be securely run in parallel with respect to the relevant adversary (i.e., with respect to either semi-honest or malicious adversaries).

3.2.4. A note on evaluating multiple polynomials with the same receiver input. The OPE protocols have a rather handy property: a *single* invocation of a protocol can be used to let the receiver compute the values of *several* polynomials at the *same* point x , with the same overhead (in terms of the number of oblivious transfers) as computing the value of a single polynomial. This can be done since the choices of the receiver in the OT protocols depend on its input x alone, and this input is the same in all invocations of the OPE protocol. The parallel invocation is done by the parties running a protocol in which the sender's input is n polynomials $\langle P_1, \dots, P_n \rangle$, the receiver's input is x , and the receiver's output is $\langle P_1(x), \dots, P_n(x) \rangle$. The sender defines appropriate polynomials $\langle Q_1, \dots, Q_n \rangle$ and in every step in which the sender transfers a single value $Q(i, j)$ in the original protocol, it transfers n values $\langle Q_1(i, j), \dots, Q_n(i, j) \rangle$ in the multipolynomial protocol. Note that this variant ensures that the receiver computes the values of all the polynomials at the *same point* x . This variant of the protocol is used in Protocol 3.4 in section 3.4, and is also used in [26] for distributed generation of RSA keys.

3.3. Security analysis for semi-honest behavior. In order to prove the security of the protocol one must show that the privacy of both parties is preserved. The privacy of the receiver is based on the pseudorandomness assumption which is used (namely, Assumption 1 or 2), while the privacy of the sender is independent of this assumption. The proof employs a hybrid model in which there is access to an ideal OT functionality, as is defined in Definition 1.5. Composition theorems (see, e.g., [29, 10, 11]) can then be used to replace the ideal OT functionality with a specific implementation of OT, as is described in section 1.3.1.

3.3.1. The receiver’s privacy. The privacy goal of the receiver is to hide the value $\alpha = S(0)$ from the sender. We only need to show this for a semi-honest sender, since the protocol includes a single message from the receiver to the sender, which does not depend on the operation of the sender (and our security definition does not require stimulation of the output of the receiver together with the view of the sender). This property is guaranteed by the pseudorandomness assumptions stated in section 2.2. We prove this result for Protocol 3.2, based on pseudorandomness Assumption 1. The proof for Protocol 3.3 is identical and is based on pseudorandomness Assumption 2.

THEOREM 3.1. *If the sender can distinguish between two different inputs of the receiver in Protocol 3.2, then pseudorandomness Assumption 1 does not hold (namely, there is a distinguisher between the two probability ensembles stated in the assumption).*

Proof. The sender’s view in the protocol contains the set of points that is given to him by the receiver, and the interaction between the two parties in the oblivious transfer protocol. Each instance of the sender’s view therefore contains nm points, and in addition his view in an oblivious transfer protocol in which the receiver chooses to learn values associated with a set of n of the nm points. Recall also that we assume that the protocol uses an ideal OT functionality.

Let α_0, α_1 be any two values in \mathcal{F} . Consider the following probability distributions of instances of the sender’s view in the protocol.

(i) $S_{OT,0}$: The set of nm points is chosen randomly from $A_{n,m}^{k,\alpha_0}$. The interaction of the receiver in the oblivious transfer protocol corresponds to it choosing to learn the n correct points of the polynomial among the nm points.

(ii) $S_{OT,1}$: Similar to $S_{OT,0}$, with the only difference being that the set of nm points is chosen randomly from $A_{n,m}^{k,\alpha_1}$.

(iii) $S_{R,0}$: The set of nm points is chosen randomly from $A_{n,m}^{k,\alpha_0}$. The interaction of the receiver in the oblivious transfer protocol corresponds to it choosing to learn n points sampled at random from the set of nm points.

(iv) $S_{R,1}$: Similar to $S_{R,0}$, with the only difference being that the set of nm points is chosen randomly from $A_{n,m}^{k,\alpha_1}$.

We would like to show that no algorithm D_{α_0,α_1} run by the sender can distinguish between an input sampled from $S_{OT,0}$ and an input sampled from $S_{OT,1}$. Define $D_{OT,0}$ as the probability that the output of D_{α_0,α_1} is 1 given an input sampled from $S_{OT,0}$. Similarly, define $D_{OT,1}$, $D_{R,0}$ and $D_{R,1}$. It holds that

$$|D_{OT,0} - D_{OT,1}| \leq |D_{OT,0} - D_{R,0}| + |D_{R,0} - D_{R,1}| + |D_{R,1} - D_{OT,1}|.$$

Assume to the contrary that $|D_{OT,0} - D_{OT,1}|$ is nonnegligible. In this case either the value $|D_{OT,0} - D_{R,0}| + |D_{R,1} - D_{OT,1}|$ is nonnegligible (option 1), or the value $|D_{R,0} - D_{R,1}|$ is nonnegligible (option 2).

If option 1 occurs, then either $|D_{R,0} - D_{OT,0}|$ or $|D_{R,1} - D_{OT,1}|$ is nonnegligible. This is a contradiction since in this case the sender can distinguish between different

inputs of the receiver to the oblivious transfer protocol. (It is possible to show a reduction that uses the fact that, e.g., $|D_{R,0} - D_{OT,0}|$ is nonnegligible, to distinguish between different inputs of the receiver.)

If option 2 occurs then we have a distinguisher between inputs sampled from $A_{n,m}^{k,\alpha_0}$ and from $A_{n,m}^{k,\alpha_1}$. The distinguisher operates by receiving its input, adding to it an interaction for the oblivious transfer protocol in which the receiver learns a random set of n values, and forwarding the combined input to D_{α_0,α_1} . Since $|D_{R,0} - D_{R,1}|$ is nonnegligible there will be a nonnegligible difference between the output being 1 given inputs from the two distributions. \square

3.3.2. The sender’s privacy—the case of a semi-honest receiver. We first assume a semi-honest receiver whose operation follows the behavior that it should take according to the protocol. This is a good model for the case of a truthful party that executes the protocol, but at a later stage falls prey to an adversary that might examine the information learned during the protocol execution. The proof for the case of a semi-honest receiver, as we show in section 3.4, is also sufficient for the case of *linear* polynomials, even if the receiver is malicious. Later we show how to provide privacy in the general case of malicious receivers.

The proof that the sender’s privacy is preserved against semi-honest receivers is identical for all protocols. The sender hides the polynomial P in a bivariate polynomial Q that is generated by adding to $P()$ a random polynomial in x , and the receiver obtains a system of $d_R + 1 = d + 1 = kd_P + 1$ values of Q , using different x values. Lemma 3.2 states that the receiver learns only a single linear combination of the coefficients of P . In particular this implies (Corollary 3.1) that a semi-honest receiver learns only a single value of the polynomial P .

LEMMA 3.2. *Let $Q(x, y)$ be a bivariate polynomial of the form*

$$Q(x, y) = P_x(x) + P(y) = \sum_{i=1}^d a_i x^i + \sum_{i=0}^{d_P} b_i y^i.$$

Then for any $d + 1$ values x_1, \dots, x_{d+1} , which are distinct and different from 0, and for any $d + 1$ values y_1, \dots, y_{d+1} , the distribution of $\{Q(x_j, y_j)\}_{j=1}^{d+1}$ is either independent of the coefficients b_0, \dots, b_{d_P} , or depends on a single linear equation of these coefficients.

Proof. The values of $Q(x_j, y_j)$ are equations of the form $Q(x_j, y_j) = \sum_{i=1}^d a_j \cdot (x_j)^i + \sum_{i=0}^{d_P} b_i \cdot (y_j)^i$. They correspond, therefore, to a set of $d + 1$ equations of the following form, with different x_j ’s:

$$\underbrace{\begin{pmatrix} (x_{j_1})^d & (x_{j_1})^{d-1} & \dots & x_{j_1} & y_{j_1}^{d_P} & \dots & 1 \\ (x_{j_2})^d & (x_{j_2})^{d-1} & \dots & x_{j_2} & y_{j_2}^{d_P} & \dots & 1 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ (x_{j_{d+1}})^d & (x_{j_{d+1}})^{d-1} & \dots & x_{j_{d+1}} & y_{j_{d+1}}^{d_P} & \dots & 1 \end{pmatrix}}_A \begin{pmatrix} a_d \\ \vdots \\ a_1 \\ b_{d_P} \\ \vdots \\ b_0 \end{pmatrix} = \begin{pmatrix} Q(x_{j_1}, y_{j_1}) \\ Q(x_{j_2}, y_{j_2}) \\ \vdots \\ Q(x_{j_{d+1}}, y_{j_{d+1}}) \end{pmatrix}.$$

It should be shown that regardless of the values of the points (x_j, y_j) , the rows of the matrix A do not span more than a single linear combination of the vectors

$$\{e_i = \underbrace{(0, \dots, 0, 1, 0, \dots, 0)}_i \mid d + 1 \leq i \leq d + d_P + 1\}.$$

The matrix A has $d + d_P + 1$ columns and $d + 1$ rows. Consider the matrix A' with $d + d_P + 1$ rows that is formed by taking the first d rows of A and appending to them the vectors $e_{d+1}, \dots, e_{d+d_P+1}$. The determinant of A' is different from 0, since its upper-left submatrix of size $d \times d$ is a Vandermonde matrix, and the lower-right submatrix of size $(d_P + 1) \times (d_P + 1)$ is an identity matrix. Therefore, the first d rows of A do not span any of $e_{d+1}, \dots, e_{d+d_P+1}$, and the matrix A that has just a single additional row cannot span more than a single linear combination of these vectors. \square

Note that we assume that the OT protocol implements an ideal functionality, and therefore the receiver (either semi-honest or malicious) learns $d + 1$ values from the set $\{Q(x_j, y_j)\}_{j=1}^{d+1}$. In the case of a semi-honest receiver these values are of the form $Q(x_j, P(x_j))$, whereas a malicious receiver can learn arbitrary values of $Q(x_j, y_j)$ subject to the constraint that the x coordinates are distinct and different from 0. Therefore Lemma 3.2 implies the following two corollaries.

COROLLARY 3.1. *A semi-honest receiver learns only a single value of the polynomial P .*

COROLLARY 3.2. *A malicious receiver learns only a single linear equation of the coefficients of the polynomial P .*

The latter corollary is used in section 3.4 to construct an OPE protocol which is secure against malicious adversaries.

3.4. Security against malicious behavior. As described in section 3.3.1, we should only consider a semi-honest server since the only message sent by the receiver is sent before it receives any information from the server. On the other hand, a malicious adversary that plays the receiver’s role might run the protocol and ask to learn values $Q(x, y)$ that do not correspond to a polynomial $S(x)$, i.e., are not of the form $Q(x, S(x))$. By doing so it might learn a linear combination of the coefficients of P which does not correspond to any value of $P(x)$. Lemma 3.2 shows that the adversary can only learn a single such linear combination. This might be sufficient for some applications, but to conform to the “ideal model” security definition, or, in general, to security against malicious behavior, the protocol must limit the information that the receiver can learn to a linear combination corresponding to a value of the polynomial P . This is achieved by the Protocol 3.4 that is described in what follows.

Improved efficiency. Another advantage of Protocol 3.4 is that its computational overhead, as measured by the number of oblivious transfers that are executed, is independent of the degree of P , and is equal to the overhead of Protocols 3.2 and 3.3 for the case of a linear polynomial P .

Before describing Protocol 3.4, we first note that if the polynomial P is linear, then Protocols 3.1, 3.2, and 3.3 allow the receiver to learn only a single value of P and no other information about the coefficients (regardless of whether the receiver is malicious or not). This is proved for Protocol 3.2 in the following lemma.

LEMMA 3.3. *When P is linear, the only information that the receiver can learn in Protocol 3.2 is a single value of $P(\cdot)$.*

Proof. Denote $P(x)$ as $P(x) = b_1x + b_0$. Corollary 3.2 implies that the receiver can only learn a single linear combination of the form $b_1 \cdot \gamma_1 + b_0 \cdot \gamma_0$, where the receiver knows γ_1 and γ_0 . If $\gamma_0 \neq 0$, then this is equivalent to the receiver learning $P(\gamma_1/\gamma_0) = b_1 \cdot (\gamma_1/\gamma_0) + b_0$.

We next claim that the adversary cannot learn any combination in which the coefficient γ_0 is 0. Consider the matrix A used in the proof of Theorem 3.2. The receiver can learn the scalar multiplication of the coefficient vector by a linear combination of

the rows of this matrix. In the case of a linear polynomial P , the matrix has $d+1$ rows and $d+2$ columns, where the last two columns correspond to the coefficients b_1 and b_0 of P . Note that the $(d+1) \times (d+1)$ matrix that is composed of the first d columns together with the last column, is a Vandermonde matrix. Therefore no linear combination of the $d+1$ rows of the original matrix can generate the row $(0, \dots, 0, 1, 0)$, which corresponds to the receiver learning a linear combination $b_1 \cdot \gamma_1 + b_0 \cdot \gamma_0$ in which the coefficient γ_0 is 0. \square

Using linearization to combat malicious receivers. Given Lemma 3.3 the major tool we use is a reduction from the OPE of a polynomial of degree z , to z OPEs of linear polynomials, due to Gilboa [27]. The reduction is stated in the following lemma, which is proven at the end of this section.

LEMMA 3.4 (see [27]). *For every polynomial P of degree z , there exist z linear polynomials P_1, \dots, P_z such that an OPE of P can be reduced to a parallel execution of an OPE of each of P_1, \dots, P_z , where all the linear polynomials are evaluated at the same point.*

We first describe Protocol 3.4 that uses this reduction to provide security against a malicious receiver. We then show that the privacy of the server is preserved and prove the lemma. Protocol 3.4 itself ensures that the receiver evaluates all linear polynomials at the same point. Furthermore, the overhead, in terms of oblivious transfers, is the same as that of a *single* OPE of a linear polynomial, since the choices of the receiver in the oblivious transfer invocations in all OPEs are the same, and therefore it is possible to use parallel executions of OPE as described in section 3.2.4.

PROTOCOL 3.4 (oblivious polynomial evaluation against malicious receivers). *The sender's input is P and the receiver's input is a polynomial α . The protocol is composed of the following steps:*

1. *The sender generates the d_P linear polynomials P_1, \dots, P_{d_P} that are used for reducing the OPE of the polynomial P to d_P OPEs of linear polynomials, by the method of Lemma 3.4.*
2. *The parties execute d_P instances of OPE in which the receiver evaluates the linear polynomials P_1, \dots, P_{d_P} at the point α , under the following constraints:*
 - (i) *The sender generates independent masking polynomials $P_{x,i}$, $1 \leq i \leq d_P$, and obtains the resulting bivariate polynomials $Q_i(x, y) = P_{x,i}(x) + P_i(y)$, one for each of the d_P OPEs. (step 1 of Protocol 3.1.)*
 - (ii) *The receiver generates a single polynomial S for use in all the OPEs (step 2 of Protocol 3.1). This step defines d_P polynomials $R_1(x) = Q_1(x, S(x)), \dots, R_{d_P}(x) = Q_{d_P}(x, S(x))$ such that for each i it holds that $R_i(0) = P_i(\alpha)$.*
 - (iii) *The receiver learns $d_R + 1$ tuples of the form $(x_j, R_1(x_j), \dots, R_{d_P}(x_j))$. These values enable it to interpolate $R_1(\alpha), \dots, R_{d_P}(\alpha)$ (steps 3 and 4 of Protocol 3.1).*
The implementation of this step is done by executing the same number of oblivious transfers as is required for a single OPE of a linear polynomial. In the protocol the receiver obtains values of all polynomials, namely, $(x_j, R_1(x_j), \dots, R_{d_P}(x_j))$, instead of a single value $(x_j, R(x_j))$.
3. *The receiver uses $P_1(\alpha), \dots, P_{d_P}(\alpha)$ to compute $P(\alpha)$ according to the method of Lemma 3.4.*

It follows from Lemma 3.4 and from the correctness of Protocol 3.1 that this protocol is correct. Namely, that it enables the receiver to compute $P(\alpha)$ for every value α . As for efficiency, we measure the computation overhead by the number of oblivious transfers that are executed. The protocol requires the same number of

oblivious transfers as is required by an OPE of a *linear* polynomial, regardless of the degree d_P .¹ It is therefore more efficient computationally than a direct OPE of the d_P degree polynomial P . The communication overhead is about $d_P + 1$ times larger than that of the OPE of a linear polynomial.

THEOREM 3.5. *Assuming the use of an ideal OT functionality, Protocol 3.4 is secure against malicious behavior. Namely,*

- (i) *The sender cannot distinguish between two different inputs of the receiver.*
- (ii) *The receiver learns only a single value of the polynomial P .*

Proof. The receiver’s privacy is an immediate corollary of the receiver’s privacy in the linear OPE protocol (Theorem 3.1), since the information that the receiver sends is the same as in a single OPE protocol where it asks to learn the value of a linear polynomial at $x = \alpha$. The sender’s privacy follows from Lemma 3.4 and from Lemma 3.3 (which guarantees the sender’s privacy against malicious receivers in the case of linear polynomials). \square

We now turn to the proof of Lemma 3.4.

Proof of Lemma 3.4. The lemma follows from the following three claims. \square

CLAIM 3.1. *For every polynomial P of degree z the server can define z linear polynomials P_1, \dots, P_z such that given $P_1(\alpha), \dots, P_z(\alpha)$, it is possible to compute the value of $P(\alpha)$.*

Proof. Denote the polynomial P as $P(x) = \sum_{i=0}^z b_i x^i$, where z denotes the degree of the polynomial. The Horner representation of this polynomial is the following:

$$P(x) = (((b_z x + b_{z-1}) \cdot x + b_{z-2}) \cdot x) + \dots + b_0.$$

The innermost linear polynomial of the Horner representation is $Q_z(x) = b_z x + b_{z-1}$. Define the linear polynomial $P_z(x) = b_z x + b_{z-1} - s_z$, where s_z is a random value chosen by the server. Of course, it holds that $P_z(x) + s_z = Q_z(x)$. Suppose that the client learns the value $P_z(\alpha)$ (say, by executing an OPE). Following this step, the client and server have two random shares, $P_z(\alpha)$ and s_z , that sum up to $Q_z(\alpha)$. Now, the innermost polynomial of degree 2 is the following:

$$Q_{z-1}(x) = Q_z(x) \cdot x + b_{z-2} = (P_z(x) + s_z) \cdot x + b_{z-2} = P_z(x) \cdot x + s_z \cdot x + b_{z-2}.$$

Define $P_{z-1}(x) = s_z \cdot x + b_{z-2} - s_{z-1}$, where s_{z-1} is randomly chosen by the server. Then,

$$Q_{z-1}(\alpha) = P_z(\alpha) \cdot \alpha + P_{z-1}(\alpha) + s_{z-1}.$$

Suppose now that the client learns $P_{z-1}(\alpha)$ (by executing an OPE). Now the parties know two random shares that sum up to $Q_{z-1}(\alpha)$: the client can compute the share $P_z(\alpha) \cdot \alpha + P_{z-1}(\alpha)$, and the server knows s_{z-1} .

In the general case, the inner polynomial of degree $z - i$ can be represented as

$$\begin{aligned} Q_i(x) &= Q_{i+1}(x) \cdot x + b_{i-1} \\ &= P_z(x) \cdot x^{z-i} + P_{z-1}(x) \cdot x^{z-i-1} + \dots + P_{i+1}(x) \cdot x + s_{i+1} \cdot x + b_{i-1}. \end{aligned}$$

¹The length of the inputs of the oblivious transfer protocols is likely to be larger than the security parameter. This fact does not increase the number of oblivious transfers: the sender encrypts the long inputs with random keys, uses oblivious transfer to let the receiver learn one of the keys, and sends all encrypted inputs to the receiver. The receiver then uses the key to decrypt one of the inputs.

The server chooses a random value s_i and defines the linear polynomial $P_i(x) = s_{i+1} \cdot x + b_{i-1} - s_i$. It now holds that $(P_z(\alpha) \cdot \alpha^{z-i} + P_{z-1}(\alpha) \cdot \alpha^{z-i-1} + \dots + P_{i+1}(\alpha) \cdot \alpha + P_i(\alpha))$ and s_i are two random shares, which can be computed by the client and the server, respectively, and which sum up to $Q_i(\alpha)$. This definition is used up to $P_2(x)$; for $P_1(x)$, the server defines $P_1(x) = s_2 \cdot x + b_0$ (without any random value m_1).

We get that $Q_1(x) = P(x)$, and therefore

$$(3.1) \quad P(\alpha) = P_z(\alpha) \cdot \alpha^{z-1} + P_{z-1}(\alpha) \cdot \alpha^{z-2} + \dots + P_1(\alpha). \quad \square$$

CLAIM 3.2. *The computation of the z linear polynomials of Claim 3.4 can be done by z OPEs that are executed in parallel.*

Proof. Note that the polynomials P_1, \dots, P_z do not depend on α . Therefore, the server can define them in advance (by defining the values s_2, \dots, s_z). This enables the parties to execute z parallel invocations of OPE, in which the client computes $P_1(\alpha), \dots, P_z(\alpha)$. At the end of this stage, the client is able to compute $P(\alpha)$ using (3.1).

It follows that for every $\alpha \in \mathcal{F}$, the client can compute $P(\alpha)$ by learning the values $P_1(\alpha), \dots, P_z(\alpha)$. If the parties use the OPE protocols described in this paper then the $z + 1$ OPEs can be implemented with the client sending the same message in all invocations, which is the same information as in a single OPE of a linear polynomial at the point α . The server responds with the relevant answer for each of the z polynomials. \square

In order to prove that the only information about P that can be computed given $P_1(\alpha), \dots, P_z(\alpha)$ is $P(\alpha)$ we prove the following claim.

CLAIM 3.3. *Given $P(\alpha)$ it is possible to simulate the client's output in the z invocations of the OPE protocols. (Namely, the only information about P that can be computed given $P_1(\alpha), \dots, P_z(\alpha)$ is $P(\alpha)$.)*

Proof. To prove the claim we show that for every $\langle P, \alpha \rangle$ it holds that the vector $\langle P_2(\alpha), \dots, P_z(\alpha) \rangle$ is uniformly distributed in \mathcal{F}^{z-1} given that $\langle s_2, \dots, s_z \rangle$ are uniformly distributed in \mathcal{F}^{z-1} .

For every $2 \leq i \leq z$, observe that we can write $P_i(\alpha) = C_i(P, \alpha, s_{i+1}, \dots, s_z) - s_i$, where $C_i(P, \alpha, s_{i+1}, \dots, s_z)$ is a function of the coefficients of P , of α , and of s_{i+1}, \dots, s_z . The claim is proved by induction, showing that $P_i(\alpha), \dots, P_z(\alpha)$ is random, with i going from z down to 2. The base case, $i = z$ is clear. In every step, we observe that s_i , which is chosen at random, is used to define $P_i(\alpha)$ alone and is not involved in defining $P_{i+1}(\alpha), \dots, P_z(\alpha)$. Therefore the vector $P_i(\alpha), \dots, P_z(\alpha)$ is randomly distributed given that $P_{i+1}(\alpha), \dots, P_z(\alpha)$ is randomly distributed.

Now, recall (3.1). $P_1(\alpha)$ is well defined given $\alpha, P(\alpha)$, and the values of $P_2(\alpha), \dots, P_z(\alpha)$. Therefore to simulate the client's view we choose random values for $P_2(\alpha), \dots, P_z(\alpha)$ and compute $P_1(\alpha)$ according to (3.1). \square

Realizing oblivious transfer. The security proofs show a reduction of security to that of an ideal OT functionality. All implementations of OT which are described in section 1.3.1 provide security against semi-honest adversaries. In the malicious case, the security provided by Protocol 3.4 depends on the security of the protocol which is used to realize the OT functionality. Universally composable OT protocols obviously result in a secure OPE protocol, since the choices of the receiver in the invocations of the OT protocol define a single point in which it can evaluate the polynomial. Invoking an information-theoretic OT protocol results in an OPE protocol which provides information-theoretic security for the sender. In this case the receiver cannot learn more than a single point of the polynomial, but it might not be possible to extract

it. Finally, it is also possible to use OT protocols that provide computational security for the sender, where the receiver's input is extractable if the OT invocations are done sequentially. In this case, however, the protocols must be invoked sequentially in order to enable extraction of the receiver's input from the OPE protocol.

4. Applications of oblivious evaluation of polynomials. There are two types of applications in which oblivious polynomial evaluation is useful. The first is where the receiver obtains a value from a k -wise independent space. For many applications such values are as good as truly random or pseudorandom values. The second type of application is of cryptographic protocols in which users must obtain a value of a polynomial held by another party without revealing their input. We give examples of both types of applications below. We also give a short description of applying the protocols of this paper to obviously computing $g^{P(x)}$, where P is a polynomial.

4.1. Obviously computing a polynomial in the exponent. In some scenarios it might be required to let the receiver compute the value of $g^{P(x)}$, where g is a generator of some group, P is known to the sender and x is known to the receiver. This computation is likely to be useful for cryptographic protocols that are based on the Diffie–Hellman assumption [19].

All the protocols described in this paper can be readily applied for this computation. The only change is for the sender to provide the receiver with values of $g^{R(x)} = g^{Q(x,S(x))}$, instead of values of $R(x) = Q(x,S(x))$. The receiver needs to interpolate these values to compute $g^{R(0)} = g^{P(\alpha)}$. She can do so by computing

$$g^{R(0)} = g \sum_{i=1}^{d_R+1} \lambda_i \cdot R(i) = \prod_{i=1}^{d_R+1} g^{\lambda_i \cdot R(i)} = \prod_{i=1}^{d_R+1} (g^{R(i)})^{\lambda_i},$$

where the values $\{\lambda_i\}_{i=1}^{d_R+1}$ are the appropriate Lagrange coefficients.

4.2. Comparing information without leaking it. Imagine two parties, A and B . Each of the parties holds a particular name (e.g., of a “suspect” from a small group of people). The two parties would like to check whether they both have the same input, under the condition that if the inputs are different they do not want to reveal any information about them (except for the fact that they are different). The main obstacle in designing a protocol for this problem is that the domain of inputs, e.g., names, is probably small enough to enable a brute force search over all possible inputs. This problem was thoroughly discussed by Fagin, Naor, and Winkler [22], with subsequent constructions by Crépeau and Salvail [16].

To specify the function more accurately, we require that if the parties' inputs are (α, β) , respectively, then their outputs are $(1, 1)$ if $\alpha = \beta$, and $(0, 0)$ otherwise. In the case of malicious parties we relax the requirement and say that while for $\alpha = \beta$ the outputs can be arbitrary (since a malicious party can always change its input or output), in the case of inputs $\alpha \neq \beta$, the output of the honest party must be 0. Namely, the malicious party cannot convince the honest party that the inputs are equal. (This requirement is natural in the context of password verification, discussed below, in which if one party does not provide the right password then the other party must reject.)

Since we cannot prevent a malicious party from choosing or guessing its input, we refer to the common security notion of simulation: for every behavior of the malicious party it should be possible to construct a simulator which has access to the trusted

third party (TTP) in the ideal model (but has no access to the input of the honest party) and can interact with the malicious party. The simulator should simulate the output distribution of the two parties, where the output of the honest party satisfies the requirement above. (In our case, the simulator will operate by extracting the input that the malicious party provides to the OPE and sending it as an input to the TTP.)

Oblivious evaluation of linear polynomials can be used to construct a very simple solution to the comparison of information problem.

PROTOCOL 4.1 (privacy preserving comparison of information).

- (i) *Input: Denote A 's input as α and B 's input as β .*
- (ii) *Party A generates a random linear polynomial $P_A(\cdot)$.*
- (iii) *Party B generates a random linear polynomial $P_B(\cdot)$.*
- (iv) *The parties execute the oblivious polynomial evaluation twice, switching roles.*

(i) *In the first invocation, A obviously learns a value of B 's polynomial (it should choose to learn $P_B(\alpha)$).*

(ii) *In the second invocation, B obviously learns a value of A 's polynomial (it should choose to learn $P_A(\beta)$).*

(iii) *The parties compute and compare the two values, $P_A(\alpha) + P_B(\alpha)$ (computed by A), and $P_A(\beta) + P_B(\beta)$ (computed by B).*

(iv) *If these values are equal then each party outputs 1. Otherwise the parties output 0. (If $\alpha = \beta$, then the two values are the same, otherwise they are different with probability $1/|\mathcal{F}|$, where \mathcal{F} is the field over which the polynomial is defined.)*

For semi-honest parties, privacy is preserved since the parties compare values of the function $P_A(x) + P_B(x)$ that has the following properties (which are trivial to prove):

- (i) The function is pairwise independent.
- (ii) Each party only computes $P_A(x) + P_B(x)$ once.
- (iii) Each party computes $P_A(x) + P_B(x)$ without revealing x to the other party.

In the case of malicious parties, the proof is simple if the protocol uses an OPE protocol which enables the extraction of the receiver's input. In this case, given a TTP which computes the function in the ideal model, we can simulate the joint distribution of the malicious party and the output of the other party: assume that Alice is malicious. We extract her input α from her invocations of the OPE protocol and provide α to the TTP. If the answer is 1, we continue the protocol by evaluating her polynomial at $\beta = \alpha$, sending the value $P_A(\alpha) + P_B(\alpha)$ to the comparison, and fixing Bob's output based on the result of the comparison. If the answer of the TTP is 0 then we provide a random value to the comparison. (Note that in the case of a malicious Bob, who computes a value of Alice's polynomial *after* letting her evaluate his, we cannot extract Bob's input before evaluating $P_B()$. We can, however, execute the OPE of Bob's polynomial twice in the simulation, learn $P_B()$ completely, and then be able to compute any value of $P_B()$ and use it to provide the right value to the comparison.)

Note that for all OPE protocols it holds that if the evaluation point does not match the input of the honest party, then with high probability the output of the protocol is 0. If the OPE protocol does not enable extraction, then we don't know, however, how to extract the evaluation point. (It might even be possible that the malicious party has some computational representation of the other party's input such that it can evaluate the OPE in the right point but does not have an explicit knowledge of the value of its input.)

Application to passwords. The protocol can serve as a basis for a mutually authenticated key exchange based on (possibly weak) passwords. Consider a user who wishes to log in to a remote server over an insecure network. She does not want to send her password in the clear, and is not even certain that the remote party is the required server. An additional problem is that the password might not have enough entropy and might therefore be susceptible to dictionary attacks. Assume that there is no PKI (public key infrastructure), and that the user does not carry with her a public key of the remote server. There have been several initial solutions to this problem (see, e.g., [5, 47]), for which it seems that a security proof must postulate the existence of a random oracle.

The most natural formulation of the scenario is as a “comparing information without leaking it” problem. If the right user contacts the right server they both should be “thinking” of the same password, and they can verify whether this is the case using Protocol 4.1. Therefore, if it is assumed that there is no active adversary, and an adversary only listens to the communication between the two parties and then tries to impersonate the user, then Protocol 4.1 can be used for password authentication. Furthermore, it can be used to generate a session key for the two parties, whose entropy does not depend on the entropy of the passwords.

In more detail, each of the parties, the user and the remote server, chooses a random linear polynomial and (obviously) computes the sum of the two polynomials at $x = \text{password}$. They use the first half of the output for authentication and the second half as a session key.

If the adversary can also be active, i.e., change the communication sent between the two parties, then the above protocol is insufficient. Although the adversary cannot decrypt the messages sent between the parties (e.g., in the invocations of the oblivious transfer protocol), it can change them and cause the output of the oblivious transfer protocol to be different, but related, to its legitimate output. The adversary can use this feature to attack different invocations of the protocol that are being executed in parallel. Our protocol can serve as a basis for a protocol that prevents such attacks, which must address delicate issues such as the nonmalleability [18] of the oblivious transfer protocols. Elaborate definitions and constructions of such protocols were given in subsequent work (see, e.g., [30, 55]).

4.3. Secure computation using randomizing polynomials. Ishai and Kushilevitz showed that secure evaluation of arbitrary functions can be reduced to secure evaluation of (perhaps several) degree-3 polynomials [36]. The inputs of these polynomials are the input variables of the original function and additional variables which must take random values. The size of the polynomials is quadratic in the size of a branching program computing the original function (therefore, this approach is efficient for functions that have an efficient branching program representation).

An examination of the structure of the polynomials generated by the constructions in [36] shows that each monomial is of the form $r_i x_j r_k$, where x_j is an original input variable and r_i and r_k are variables which must be assigned random values. Each of the variables is assigned to a party that must instantiate it with its corresponding input value (in the case of x_j) or with a random value (in the case of r_j and r_k). In the case of secure two-party computation, the computation of the randomizing polynomials can be easily reduced to OPE of *linear* polynomials. Namely, the sender assigns values to each of the variables that it controls and consequently each monomial becomes either a constant or a linear polynomial of an input known to the receiver. The two parties then do a shared output OPE for each of these polynomials and compute the sum of all the shares they learned.

The original work of [36] reduced the computation of the OPE to generic secure computation (the focus of that work was on information-theoretic security). However, a more efficient solution could use the linear OPE described in this paper. Furthermore, this results in a strongly polynomial secure computation protocol for a more general class of functions—namely, functions with efficient branching program representation.

4.4. Anonymous initialization for metering. A scheme for efficient and secure metering of clients' visits to servers was suggested in [50]. This scheme involves *clients* which send requests to *servers*, and a trusted, off-line, *audit authority*. Servers prove to the audit authority that they fulfilled requests (e.g., served web pages) of a certain number of clients. On a very high level, the operation of the scheme is as follows:

(i) The audit authority generates a random bivariate polynomial $P(x, y)$, where the degree of x is $k - 1$.

(ii) Each client u receives from the audit authority initialization data, which contains a univariate polynomial $P(u, \cdot)$.

(iii) When u sends a request to a server S (e.g., visits its web site), it sends it the value $P(u, S)$.

(iv) After k requests of different clients, S can interpolate the value $P(0, S)$ which serves as a proof for serving k requests.

A modification of this scheme described in [50] preserves the anonymity of clients towards servers. However, if the audit agency cooperates with a server S they are able to identify clients which access the server, since the audit agency knows which polynomial is owned by every client. Combining this scheme with oblivious polynomial evaluation enables the client to learn the initialization data from the audit agency without revealing to the agency which initialization value (i.e., polynomial) was learned by the client. If this method is employed, then even a coalition of the audit agency and a server is not able to identify the client.

4.5. Anonymous coupons. Consider the following scenario. An organization wants to set up an *anonymous* complaint box for its personnel, but would like to ensure that each person complains at most once (for example, if there are ten complaints about the quality of the coffee machine, they should be from ten different people and not ten complaints of the same person). A solution for this problem is to give each person an anonymous coupon that he or she should attach to any submitted complaint. When a complaint is received the coupon is checked for validity and freshness (namely, it is verified that it is a valid coupon that was not previously used).

Anonymous coupons can be constructed using oblivious polynomial evaluation.² The coupon manager sets up a polynomial P of degree d . Each person A obtains a coupon by choosing a random secret value R_A and obliviously computing $P(R_A)$, using an oblivious polynomial evaluation protocol in which the coupon manager is the sender. The coupon is the pair $\langle R_A, P(R_A) \rangle$. It is easy for the coupon manager to verify that a coupon $\langle X, Y \rangle$ is valid by simply testing whether $Y = P(X)$. The coupon manager also keeps a list of the coupons that were received and compares each new coupon to that list in order to verify that it was not used before.

²Another construction for this problem can be based on using blind signatures: Each user prepares in advance coupons by letting the central server sign them using a blind signature scheme (e.g., Chaum's scheme [12], or the scheme of Juels et al. that can be based on standard hardness assumptions [37]). A complaint must be accompanied by a signed coupon. The server verifies the signature, and also verifies that this coupon has not been used before.

The system is secure against a coalition of corrupt users that attempt to generate fake coupons, as long as the coalition has at most d different coupons. The degree of the polynomial, d , should therefore be set in accordance with the potential size of a corrupt coalition of users.

A corrupt coupon manager might attempt to identify the owners of coupons by using a different polynomial for each person. Namely, for every user A it might use a different polynomial P_A for the oblivious evaluation protocol. When it later receives a coupon $\langle X, Y \rangle$ it attempts to identify its owner by checking for which polynomials P_A it holds that $Y = P_A(X)$. This attack can be prevented by ensuring that the sender uses the same polynomial in all oblivious evaluation protocols, for example, using the verifiable secret sharing techniques of Feldman [23] or Pedersen [57].

Acknowledgments. We would like to thank Sanjeev Arora, Daniel Bleichenbacher, Dan Boneh, Oded Goldreich, Yuval Ishai, Amit Klein, Ronitt Rubinfeld, Madhu Sudan, and the anonymous referees that reviewed this paper.

REFERENCES

- [1] B. AIELLO, Y. ISHAI, AND O. REINGOLD, in *Priced oblivious transfer: How to sell digital goods*, Advances in Cryptology–Eurocrypt 2001, LNCS 2045, Springer-Verlag, Berlin, 2001, pp. 119–135.
- [2] D. BEAVER, *Correlated pseudorandomness and the complexity of private computations*, in Proceedings of the 28th ACM Symposium on the Theory of Computing, 1996, pp. 479–488.
- [3] D. BEAVER AND J. FEIGENBAUM, *Hiding instances in multioracle queries*, in Proceedings of the 7th Annual Symposium on Theoretical Aspects of Computer Science, 1990, pp. 37–48.
- [4] M. BELLARE AND S. MICALI, *Non-interactive oblivious transfer and applications*, in Advances in Cryptology–CRYPTO 1989, Springer-Verlag, LNCS 435, 1990, pp. 547–557.
- [5] S. M. BELLOVIN AND M. MERRITT, *Encrypted key exchange: Password-based protocols secure against dictionary attacks*, in Proceedings of the 1992 IEEE Computer Society Conference on Research in Security and Privacy, 1992, pp. 72–84.
- [6] D. BLEICHENBACHER AND P. NGUYEN, *Noisy polynomial interpolation and noisy Chinese remaindering*, in Advances in Cryptology–EUROCRYPT 2000, Springer-Verlag, LNCS 1807, pp. 53–69.
- [7] D. BONEH, *Finding smooth integers in short intervals using CRT decoding*, J. Comput. System Sci., 64 (2002), pp. 768–784.
- [8] G. BRASSARD, C. CRÉPEAU, AND J.-M. ROBERT, *Information theoretic reduction among disclosure problems*, in Proceedings of the 27th FOCS, IEEE, 1986, pp. 168–173.
- [9] G. BRASSARD, C. CRÉPEAU, AND J.-M. ROBERT, *All-or-nothing disclosure of secrets*, in Advances in Cryptology–CRYPTO 1986, LNCS 263, Springer Verlag, 1987, pp. 234–238.
- [10] R. CANETTI, *Security and composition of multiparty cryptographic protocols*, J. Cryptology, 13 (2000), pp. 143–202.
- [11] R. CANETTI, Y. LINDELL, R. OSTROVSKY, AND A. SAHAI, *Universally composable two party computation*, in Proceedings of the 34th ACM Symposium on the Theory of Computing, 2002, pp. 494–503.
- [12] D. CHAUM, *Blind signatures for untraceable payments*, CRYPTO 1982, Plenum Press, New York, pp. 199–203.
- [13] B. CHOR, O. GOLDBREICH, E. KUSHILEVITZ, AND M. SUDAN, *Private information retrieval*, J. ACM, 45 (1998), pp. 965–981. Earlier version appeared in Proceedings of the 36th FOCS, IEEE, 1995, pp. 41–50.
- [14] D. COPPERSMITH AND M. SUDAN, *Reconstructing curves in three (and higher) dimensional space from noisy data*, in Proceedings of the 35th ACM Symposium on the Theory of Computing, 2003, pp. 136–142.
- [15] C. CRÉPEAU AND J. KILIAN, *Achieving oblivious transfer using weakened security assumptions*, in Proceedings of the 29th FOCS, IEEE, 1988, pp. 42–52.
- [16] C. CRÉPEAU AND L. SALVAIL, *Oblivious verification of common string*, CWI Quarterly, special issue for the Crypto Course 10th Anniversary, 8 (1995), pp. 97–109.
- [17] Y. DODIS AND S. MICALI, *Lower bounds for oblivious transfer reductions*, Advances in Cryptology–EUROCRYPT 1999, LNCS, Springer-Verlag, 1999, pp. 42–55.

- [18] D. DOLEV, C. DWORK, AND M. NAOR, Non-malleable cryptography, *SIAM J. Comput.*, 30 (2000), pp. 391–437. Earlier version appeared in *Proceedings of the 23rd ACM Symposium on Theory of Computing*, 1991, pp. 542–552.
- [19] W. DIFFIE AND M. HELLMAN, *New directions in cryptography*, *IEEE Trans. Inform. Theory*, 22 (1976), pp. 644–654.
- [20] P. ELIAS, *List decoding for noisy channels*, TR 335, Research Laboratory for Electronics, MIT, Cambridge, MA, 1957.
- [21] S. EVEN, O. GOLDREICH, AND A. LEMPEL, *A randomized protocol for signing contracts*, *Commun. of ACM* 28 (1985), pp. 637–647.
- [22] R. FAGIN, M. NAOR, AND P. WINKLER, *Comparing information without leaking it*, *Commun. of ACM* 39 (1996), pp. 77–85.
- [23] P. FELDMAN, *A practical scheme for non-interactive verifiable secret sharing*, in *Proceedings of the 28th FOCS*, IEEE, 1987, pp. 427–437.
- [24] J. GARAY AND P. MACKENZIE, *Concurrent oblivious transfer*, in *Proceedings of the 41st FOCS*, IEEE, 2000, pp. 314–324.
- [25] J. GARAY, P. MACKENZIE, AND K. YANG, *Efficient and universally composable committed oblivious transfer and applications*, in *Proceedings of the 1st Theory of Cryptography Conference*, LNCS 2951, Springer-Verlag, 2004, pp. 297–316.
- [26] N. GILBOA, *Two party RSA key generation*, in *Advances in Cryptology–CRYPTO 1999*, LNCS 1666, Springer-Verlag, Berlin, 1999, pp. 116–129.
- [27] N. GILBOA, *Topics in private information retrieval*, DSc. dissertation, Technion - Israel Institute of Technology, Haifa, Israel, 2000.
- [28] O. GOLDREICH, *The Foundations of Cryptography Basic Tools*, Cambridge University Press, New York, 2001.
- [29] O. GOLDREICH, *The Foundations of Cryptography - Basic Applications*, Cambridge University Press, New York, 2004.
- [30] O. GOLDREICH AND Y. LINDELL, *Session key generation using human passwords only*, in *Advances in Cryptology–CRYPTO 2001*, LNCS 2139, Springer-Verlag, Berlin, 2001, pp. 408–432.
- [31] O. GOLDREICH, S. MICALI, AND A. WIGDERSON, *How To Play Any Mental Game*, in *Proceedings of the 19th ACM Conference on Theory of Computing*, New York, New York, 1987, pp. 218–229.
- [32] O. GOLDREICH, M. SUDAN, AND R. RUBINFELD, *Learning polynomials with queries: The highly noisy case*, in *Proceedings of the 36th FOCS*, IEEE, 1995, pp. 294–303.
- [33] O. GOLDREICH AND R. VAINISH, *How to solve any protocol problem - an efficiency improvement*, in *Advances in Cryptology–CRYPTO 1987*, LNCS 293, Springer-Verlag, Berlin, 1988, pp. 297–316.
- [34] V. GURUSWAMI AND M. SUDAN, *Improved decoding of Reed-Solomon and algebraic-geometric codes*, *IEEE Trans. Inform. Theory*, 45 (1999), pp. 1757–1767. Earlier version appeared in *39th FOCS*, 1998.
- [35] Y. ISHAI, J. KILIAN, K. NISSIM, AND E. PETRANK, *Extending oblivious transfers efficiently*, in *Advances in Cryptology–CRYPTO 2004*, LNCS 2729, Springer-Verlag, Berlin, 2003, pp. 145–161.
- [36] Y. ISHAI AND E. KUSHILEVITZ, *Randomizing polynomials: A new representation with applications to round-efficient secure computation*, in *Proceedings of the 41st FOCS*, IEEE, 2000, pp. 294–304.
- [37] A. JUELS, M. LUBY, AND R. OSTROVSKY, *Security of blind signatures*, in *Advances in Cryptology–CRYPTO 1997*, LNCS 1294, Springer-Verlag, Berlin, 1997, pp. 150–164.
- [38] R. IMPAGLIAZZO AND S. RUDICH, *Limits on the provable consequences of one-way permutations*, in *Proceedings of the ACM Symposium on the Theory of Computing*, 1989, pp. 44–61.
- [39] Y. T. KALAI, *Smooth projective hashing and two-message oblivious transfer*, in *Advances in Cryptology–EUROCRYPT 2005*, Springer-Verlag, LNCS 3494, 2005, pp. 78–95.
- [40] A. KIAYIAS AND M. YUNG, *Cryptographic hardness based on the decoding of reed-solomon codes*, in *Proceedings of ICALP 2002*, LNCS 2380, Springer-Verlag, Berlin, 2002, pp. 232–243.
- [41] A. KIAYIAS AND M. YUNG, *Directions in polynomial reconstruction based cryptography (invited survey)*, *IEICE Trans.*, E87-A (2004), pp. 978–985.
- [42] J. KILIAN, *Founding cryptography on oblivious transfer*, 20th STOC, (1988), pp. 20–31.
- [43] E. KUSHILEVITZ AND R. OSTROVSKY, *Replication is not needed: single database, computationally-private information retrieval*, in *Proceedings of the 38th FOCS*, IEEE, 1997, pp. 364–373.
- [44] A. K. LENSTRA, H. W. LENSTRA, AND L. LOVASZ, *Factoring polynomials with rational coefficients*, *Mathematische Ann.*, 261 (1982), pp. 513–534.

- [45] Y. LINDELL AND B. PINKAS, *Privacy preserving data mining*, J. Cryptology, 15 (2002), pp. 177–206.
- [46] R. J. LIPTON, *Efficient checking of computations*, in Proceedings of STACS '90, 1990, pp. 207–215.
- [47] S. LUCKS, *Open key exchange: How to defeat dictionary attacks without encrypting public keys*, in Proceedings of Security Protocol Workshop '97, LNCS 1361, Springer-Verlag, Berlin, 1997, pp. 79–90.
- [48] F. J. MACWILLIAMS AND N. SLOANE, *The Theory of Error Correcting Codes*, North-Holland, Amsterdam, 1977.
- [49] M. NAOR AND K. NISSIM, *Communication preserving protocols for secure function evaluation*, in Proceedings of the 33rd ACM Symposium on the Theory of Computing, 2001, pp. 590–599.
- [50] M. NAOR AND B. PINKAS, *Secure and efficient metering*, in Advances in Cryptology—EUROCRYPT 1998, Springer-Verlag, LNCS 1403, 1998, pp. 576–590.
- [51] M. NAOR AND B. PINKAS, *Oblivious transfer and polynomial evaluation*, in Proceedings of the 31st ACM Symposium on the Theory of Computing, 1999, pp. 245–254.
- [52] M. NAOR AND B. PINKAS, *Efficient oblivious transfer protocols*, in Proceedings of the SIAM Symposium on Discrete Algorithms, 2001, pp. 448–457.
- [53] M. NAOR AND B. PINKAS, *Computationally secure oblivious transfer*, J. Cryptology, 18 (2005), pp. 1–35.
- [54] M. NAOR AND O. REINGOLD, *Number-theoretic constructions of efficient pseudo-random functions*, in Proceedings of the 38th FOCS, IEEE, 1997, pp. 458–467.
- [55] M.-H. NGUYEN AND S. VADHAN, *Simpler session-key generation from short random passwords*, in Proceedings of the Theory of Cryptography Conference, LNCS 2951, Springer-Verlag, Berlin, 2004, pp. 428–445.
- [56] P. PAILLIER, *Public-key cryptosystems based on composite degree residuosity classes*, in *Advances in Cryptology—EUROCRYPT '99*, LNCS 1592, Springer-Verlag, (1999), pp. 223–238.
- [57] T. P. PEDERSEN, *Non-interactive and information-theoretic secure verifiable secret sharing*, Crypto '91, LNCS 576, (1991), pp. 129–140.
- [58] M. O. RABIN, *How to exchange secrets by oblivious transfer*, Technical memo TR-81, Aiken Computation Laboratory, Harvard University, Cambridge, MA, 1981.
- [59] M. SUDAN, *Decoding of Reed Solomon codes beyond the error-correction diameter*, J. Complexity, 13 (1997), pp. 180–193.
- [60] A. C. YAO, *How to generate and exchange secrets*, in Proceedings of the 27th FOCS, IEEE, 1986, pp. 162–167.

AN ALGORITHM FOR COMPUTING FUNDAMENTAL SOLUTIONS*

KLAUS WEIHRAUCH[†] AND NING ZHONG[‡]

Abstract. For a partial differential operator $P = \sum_{|\alpha| \leq m} c_\alpha D^\alpha$ with constant coefficients, a generalized function u is a fundamental solution if $Pu = \delta$, where δ is the Dirac distribution. In this article, we provide an algorithm which computes a fundamental solution for every such differential operator P on a Turing machine if the input- and output-data are represented canonically.

Key words. computable analysis, partial differential equations, fundamental solution

AMS subject classifications. 03D80, 35D05, 68Q99

DOI. 10.1137/S0097539704446360

1. Introduction. In the theory of differential operators with constant coefficients, fundamental solutions have a central place. Let

$$(1) \quad P = P(D) = \sum_{|\alpha| \leq m} c_\alpha D^\alpha$$

be a partial differential operator with constant coefficients c_α , where $\alpha = (\alpha_1, \dots, \alpha_n)$ is a multi-index of order $|\alpha| = \alpha_1 + \dots + \alpha_n$, $D_j = (1/i)\partial/\partial x_j$ with $1 \leq j \leq n$ and $i = \sqrt{-1}$, $D = (D_1, \dots, D_n)$, and $D^\alpha = D_1^{\alpha_1} D_2^{\alpha_2} \dots D_n^{\alpha_n}$. A distribution is called a fundamental solution of the partial differential operator P if it is a solution of the point source problem

$$(2) \quad P(D)u = \sum_{|\alpha| \leq m} c_\alpha D^\alpha u = \delta,$$

where δ is the Dirac measure at 0. In 1954 Malgrange and Ehrenpreis proved that every partial differential operator with constant coefficients has a fundamental solution. Fundamental solutions are very useful tools in the theory of partial differential equations, for instance, in solving inhomogeneous equations and in providing information about the regularity and growth of solutions. In the case of solving inhomogeneous equations, if E is a fundamental solution of the partial differential operator $P = P(D)$ and if f is a distribution, then $E * f$ is a solution of the equation $P(D)u = f$ whenever the convolution is defined.

Many classical differential operators are known to have computable functions as fundamental solutions. For example, the Schwartz function

$$E_H(t, x) = (4\pi\nu t)^{-n/2} e^{-|x|^2/4\nu t},$$

which is a computable real function, is a fundamental solution of the heat equation

$$\frac{\partial u}{\partial t} = \nu \sum_{i=1}^n \frac{\partial^2 u}{\partial x_i^2}, \quad (t, x) \in \mathbb{R} \times \mathbb{R}^n.$$

*Received by the editors October 25, 2004; accepted for publication (in revised form) September 22, 2005; published electronically March 24, 2006. A preliminary version of this work appeared in [5].

<http://www.siam.org/journals/sicomp/35-6/44636.html>

[†]FernUniversität Hagen, D-58084 Hagen, Germany (Klaus.Weihrauch@FernUni-Hagen.de).

[‡]Department of Mathematical Sciences, University of Cincinnati, Cincinnati, OH 45221-0025 (Ning.Zhong@uc.edu). This author was supported in part by the University of Cincinnati's Summer Faculty Research Fellowship.

In general, however, a fundamental solution is a distribution. Does every partial differential operator $P(D)$ with computable coefficients have a computable fundamental solution? In fact, this follows from the even more general result we prove in this note: There is a computable operator mapping every differential operator $P(D)$ (as in (1)) to a fundamental solution. We present an algorithm which in a well-defined realistic model of computation computes a fundamental solution of any given differential operator P from its coefficients, where abstract data are encoded canonically by, generally infinite, sequences of symbols and where computations are (can be) performed by Turing machines.

2. Preliminaries. In this note, we consider the representation approach as a model of computation for analysis [4]. Computable functions on Σ^* and Σ^ω (the set of finite and infinite sequences of symbols, respectively, from the finite alphabet Σ) are defined by Turing machines which can read and write finite and infinite sequences. A multifunction $f : \subseteq X \rightrightarrows Y$ is a function which assigns to every $x \in X$ a set $f(x) \subseteq Y$, the set of “acceptable” values ($f(x) = \emptyset$ if $x \notin \text{dom}(f)$). Any concrete computation will produce on input $a \in \text{dom}(f)$ some element $b \in f(a)$, but usually there is no method to select a specific one.

Computability on other sets is defined by using Σ^* and Σ^ω as codes or names. In some of our applications, a “name” $w \in \Sigma^*$ or $p \in \Sigma^\omega$ contains information about a point x , which is sufficient for certain computations but may not identify x . Therefore, we use multifunctions as naming systems. A *naming system* of a set M is a surjective multifunction $\nu : \subseteq \Sigma^* \rightrightarrows M$ (a *notation*) or $\gamma : \subseteq \Sigma^\omega \rightrightarrows M$ (a *representation*). For the natural numbers and the rational numbers we use canonical notations $\nu_{\mathbb{N}} : \subseteq \Sigma^* \rightarrow \mathbb{N}$ and $\nu_{\mathbb{Q}} : \subseteq \Sigma^* \rightarrow \mathbb{Q}$, respectively. For the real numbers we use the representation $\rho : \subseteq \Sigma^\omega \rightarrow \mathbb{R}$, where $\rho(p) = x$ if p encodes a sequence $(a_i)_i$ of rational numbers such that $|x - a_i| \leq 2^{-i}$. For a naming system $\gamma : \subseteq Y \rightarrow M$ ($Y \in \{\Sigma^*, \Sigma^\omega\}$) the representation $\gamma^k : \subseteq Y \rightarrow M^k$ is defined by $\gamma^k \langle y_1, \dots, y_k \rangle := (\gamma(y_1), \dots, \gamma(y_k))$, where $\langle \rangle$ is a tupling function. For the complex numbers we use the representation ρ^2 of \mathbb{R}^2 .

For naming systems $\gamma_i : \subseteq Y_i \rightrightarrows M_i$ a function $h : \subseteq Y_1 \rightarrow Y_2$ (on names) realizes a multifunction $f : \subseteq M_1 \rightrightarrows M_2$ if

$$\gamma_2 \circ h(p) \cap f(x) \neq \emptyset \quad \text{if } \gamma_1(p) \in \text{dom}(f);$$

that is, $h(p)$ is a name of some $y \in f(x)$ if p is a name of $x \in \text{dom}(f)$; see Figure 1.

The multifunction f is called (γ_1, γ_2) -computable if it has a computable realization.

Multifunctions occur naturally in computable analysis. As an example, there is an algorithm which maps every ρ -name p of $x \in \mathbb{R}$ (i.e., every sequence of rational numbers rapidly converging to x) to some $n \in \mathbb{N}$ such that $x < n$. This algorithm, however, might give another upper bound n' of x if fed with another ρ -name p' of x . The algorithm is not “ $(\rho, \nu_{\mathbb{N}})$ -extensional.” Nevertheless, the algorithm realizes the multifunction $f : \mathbb{R} \rightrightarrows \mathbb{N}$, defined by $n \in f(x) \iff x < n$. There is no $(\rho, \nu_{\mathbb{N}})$ -computable function $g : \mathbb{R} \rightarrow \mathbb{N}$ such that $x < g(x)$.

As generalizations of the “acceptable Gödel numbering $\varphi : \mathbb{N} \rightarrow P^{(1)}$ ” of the partial recursive functions, for any $a, b \in \{*, \omega\}$ there is a canonical representation $\eta^{ab} : \Sigma^\omega \rightarrow F^{ab}$ of continuous functions $f : \subseteq \Sigma^a \rightarrow \Sigma^b$ satisfying the “utm-theorem” and the “smn-theorem” [4] (abbreviation: $\eta_p^{ab} := \eta^{ab}(p)$). For naming systems $\gamma_1 : \subseteq \Sigma^a \rightarrow M_1$ and $\gamma_2 : \subseteq \Sigma^b \rightarrow M_2$ a *representation* $[\gamma_1 \rightarrow \gamma_2]$ of the *total* (γ_1, γ_2) -

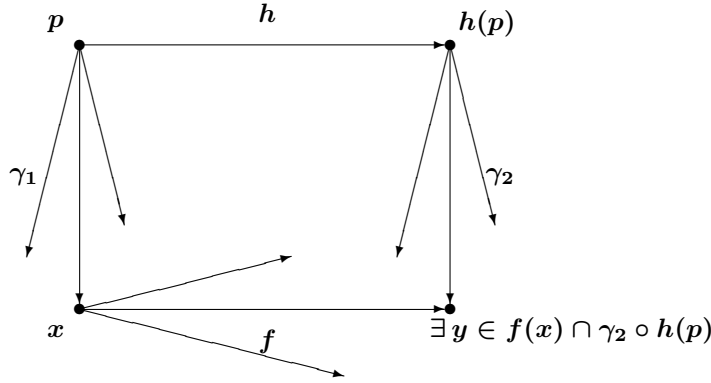


FIG. 1. $h(p)$ is a name of some $y \in f(x)$ if p is a name of $x \in \text{dom}(f)$.

continuous functions is defined by

$$[\gamma_1 \rightarrow \gamma_2](p) = f : \iff \eta_p^{ab} \text{ realizes } f \text{ w.r.t. } (\gamma_1, \gamma_2),$$

and a *multirepresentation* $[\gamma_1 \rightarrow_p \gamma_2]$ of all partial (γ_1, γ_2) -continuous functions is defined by

$$f \in [\gamma_1 \rightarrow_p \gamma_2](p) : \iff \eta_p^{ab} \text{ realizes } f \text{ w.r.t. } (\gamma_1, \gamma_2).$$

Notice that $[\gamma_1 \rightarrow \gamma_2]$ is the “weakest” representation δ of the set of total (γ_1, γ_2) -continuous functions such that the evaluation $(f, x) \mapsto f(x)$ becomes $(\delta, \gamma_1, \gamma_2)$ -computable (Lemma 3.3.14 in [4]).

We will apply the following slight generalization of the type conversion theorem in [4].

LEMMA 2.1 (type conversion). *Let $\gamma_i : \subseteq Y_i \rightarrow M_i$ be representations ($i = 1, 2, 3$), let $f : \subseteq M_1 \times M_2 \rightarrow M_3$, and let $\tilde{f} : M_1 \rightarrow \tilde{M}$ be the total function to the partial (γ_2, γ_2) -continuous functions $h : \subseteq M_2 \rightarrow M_3$ defined by $\tilde{f}(x)(y) := f(x, y)$. Then*

$$f \text{ is } (\gamma_1, \gamma_2, \gamma_3)\text{-computable} \iff \tilde{f} \text{ is } (\gamma_1, [\gamma_2 \rightarrow_p \gamma_3])\text{-computable}.$$

For definitions and mathematical properties of distributions, see [1, 2]. In [6] computability on distributions over the real line is studied. The definitions and theorems can be generalized straightforwardly to distributions over \mathbb{R}^n ($n \geq 1$). For the space of continuous functions $f : \mathbb{R}^n \rightarrow \mathbb{C}$ we use the representation $[\rho^n \rightarrow \rho^2] : \subseteq \Sigma^\omega \rightarrow C(\mathbb{R}^n)$. For the space $C^\infty(\mathbb{R}^n)$ of infinitely differentiable functions $f : \mathbb{R}^n \rightarrow \mathbb{C}$ we use the representation $\delta_\infty : \subseteq \Sigma^\omega \rightarrow C^\infty(\mathbb{R}^n)$ defined by

$$\delta_\infty(\langle p_\alpha \rangle_{\alpha \in \mathbb{N}^n}) = f : \iff (\forall \alpha \in \mathbb{N}^n) D^\alpha f = [\rho^n \rightarrow \rho^2](p_\alpha)$$

(where $\langle p_\alpha \rangle_{\alpha \in \mathbb{N}^n} \in \Sigma^\omega$ is a canonical merging of the infinitely many infinite sequences $p_\alpha \in \Sigma^\omega$, $\alpha \in \mathbb{N}^n$). The topology of $C^\infty(\mathbb{R}^n)$ can be defined by the seminorms $f \mapsto |D^m f|_k = \sup_{|\alpha| \leq m} (\sup_{x \in K} |D^\alpha f(x)|)$ as m varies over the set of nonnegative integers and K varies over the family of compact subsets of \mathbb{R}^n .

A test function is a function $f \in C^\infty(\mathbb{R}^n)$ with compact support $\text{supp}(f) := \text{cls}\{x \in \mathbb{R}^n \mid f(x) \neq 0\}$. The set of test functions is denoted by $\mathcal{D}(\mathbb{R}^n)$, and its

topology is induced by $C^\infty(\mathbb{R}^n)$. We use the representation $\delta_D : \subseteq \Sigma^\omega \rightarrow \mathcal{D}(\mathbb{R}^n)$ defined by

$$\delta_D(0^k 1p) = f : \iff \delta_\infty(p) = f \text{ and } \text{supp}(f) \subseteq [-k; k]^n.$$

A Schwartz function is a function $f \in C^\infty(\mathbb{R}^n)$ such that $\sup_{x \in \mathbb{R}^n} (|x|^j |D^\alpha f(x)|) < \infty$ for all $j \in \mathbb{N}$ and $\alpha \in \mathbb{N}^n$. The set of Schwartz functions is denoted by $\mathcal{S}(\mathbb{R}^n)$. We use the representation $\delta_S : \subseteq \Sigma^\omega \rightarrow \mathcal{S}(\mathbb{R}^n)$ defined by

$$\delta_S(\langle p, q \rangle) = f : \iff \delta_\infty(p) = f \text{ and } q \text{ encodes a function } h : \mathbb{N}^{n+1} \rightarrow \mathbb{N} \text{ such that } (\forall j, \alpha) \sup_{x \in \mathbb{R}^n} (|x|^j |D^\alpha f|) \leq h(j, \alpha).$$

Of course, the above representations can be replaced by equivalent ones. Examples for the case $n = 1$ are discussed in [6].

A continuous linear map $T : \mathcal{D}(\mathbb{R}^n) \rightarrow \mathbb{C}$ (or $T : \mathcal{S}(\mathbb{R}^n) \rightarrow \mathbb{C}$) is called a distribution (tempered distribution). The set of all distributions (tempered distributions) is denoted as $\mathcal{D}'(\mathbb{R}^n)$ ($\mathcal{S}'(\mathbb{R}^n)$). For the set $\mathcal{D}'(\mathbb{R}^n)$ we use the representation $[\delta_D \rightarrow \rho^2]$ (restricted to the linear functions).

Recall that the evaluation function is computable; i.e., there is Type-2 Turing machine which computes a ρ^2 -name of $T(\phi)$ whenever given a $[\delta_D \rightarrow \rho^2]$ -name of T and a δ_D -name of ϕ as input. Usually $T(\phi)$ is written as $\langle T, \phi \rangle$.

We conclude this section by recalling some further definitions and facts [2]. The Dirac measure δ on \mathbb{R}^n at 0 is a tempered distribution defined by $\delta(\phi) = \langle \delta, \phi \rangle = \phi(0)$ for any Schwartz function $\phi \in \mathcal{S}(\mathbb{R}^n)$. The Dirac measure can be viewed as a point source. The Fourier transform of a Schwartz function $\phi \in \mathcal{S}(\mathbb{R}^n)$, denoted by $\hat{\phi}$ or $\mathcal{F}(\phi)$, is defined as

$$\mathcal{F}(\phi)(\xi) = \hat{\phi}(\xi) := (2\pi)^{-n/2} \int_{\mathbb{R}^n} e^{-ix\xi} \phi(x) dx,$$

where $x = (x_1, x_2, \dots, x_n)$, $\xi = (\xi_1, \xi_2, \dots, \xi_n)$, and $x\xi = x_1\xi_1 + x_2\xi_2 + \dots + x_n\xi_n$. The Fourier transform is a linear bijection of $\mathcal{S}(\mathbb{R}^n)$ to itself. For every partial derivative,

$$(3) \quad \mathcal{F}(D_j \phi) = \xi_j \mathcal{F}(\phi) \text{ and } D_j \mathcal{F}(\phi) = \mathcal{F}(-\xi_j \phi).$$

Therefore, for any partial differential operator $P(D)$ with constant coefficients

$$(4) \quad \langle \mathcal{F}(P(D)), \phi \rangle = P(\xi) \hat{\phi} \quad \forall \phi \in \mathcal{S}(\mathbb{R}^n).$$

Since $\mathcal{S}(\mathbb{R}^n)$ is sequentially dense in $\mathcal{S}'(\mathbb{R}^n)$, by a duality argument, \mathcal{F} and D_j can be uniquely extended from $\mathcal{S}(\mathbb{R}^n)$ to $\mathcal{S}'(\mathbb{R}^n)$ defined by the following formulae:

$$(5) \quad \langle \mathcal{F}T, \phi \rangle := \langle T, \mathcal{F}\phi \rangle,$$

$$(6) \quad \langle D_j T, \phi \rangle := \langle T, -D_j \phi \rangle$$

for any $T \in \mathcal{S}'(\mathbb{R}^n)$ and $\phi \in \mathcal{S}(\mathbb{R}^n)$. The Fourier transform $\hat{\delta}$ of the Dirac measure δ is a constant:

$$(7) \quad \mathcal{F}(\delta) = (2\pi)^{-n/2}.$$

Let \mathcal{R} be the reflexion operator on $\mathcal{S}(\mathbb{R}^n)$ defined by $\mathcal{R}(u)(x) := u(-x)$.

LEMMA 2.2. For any $E \in \mathcal{D}'(\mathbb{R}^n)$ let $\langle \tilde{E}, \phi \rangle := \langle E, \mathcal{R}\phi \rangle$. Then E is a fundamental solution of $P(D)$; that is, $P(D)E = \delta$ iff

$$\langle \tilde{E}, P(D)\phi \rangle = \phi(0) \quad \forall \phi \in \mathcal{D}(\mathbb{R}^n).$$

Proof. By (6), $\langle E, \mathcal{R}D_j\phi \rangle = \langle E, -D_j\mathcal{R}\phi \rangle = \langle D_jE, \mathcal{R}\phi \rangle$, and therefore,

$$\langle \tilde{E}, P(D)\phi \rangle = \langle E, \mathcal{R}P(D)\phi \rangle = \langle P(D)E, \mathcal{R}\phi \rangle.$$

We obtain $P(D)E = \delta$ iff $(\forall \phi)\langle P(D)E, \mathcal{R}\phi \rangle = \mathcal{R}\phi(0) = \phi(0)$ iff

$$(\forall \phi)\langle \tilde{E}, P(D)\phi \rangle = \phi(0). \quad \square$$

3. Computing a fundamental solution. Formally, the problem of constructing a fundamental solution of a partial differential operator $P(D)$ is very easy. Indeed, suppose that we have

$$P(D)E = \delta.$$

Since $\mathcal{F}D_jT = \xi_j\mathcal{F}T$ (from (3), (5), and (6)), by taking the Fourier transform we obtain $P(\xi)\mathcal{F}E = (2\pi)^{-n/2}$, hence

$$\mathcal{F}E = \frac{(2\pi)^{-n/2}}{P(\xi)},$$

and E should be defined as the inverse Fourier transform of $(2\pi)^{-n/2}/P(\xi)$. This is meaningful if $P(\xi)$ has no real zeros. In the general case, we will overcome the difficulty by selecting domains of integration which avoid the zeros of $P(\xi)$.

For $m \geq 1$ let $\mathcal{P}(m)$ be the linear space of all polynomials $P(\xi) = \sum_{|\alpha| \leq m} c_\alpha \xi^\alpha$, $c_\alpha \in \mathbb{C}$, in n variables of degree $\leq m$, where $\xi^\alpha = \xi_1^{\alpha_1} \cdots \xi_n^{\alpha_n}$ for $\alpha = (\alpha_1, \dots, \alpha_n)$ and $|\alpha| = \alpha_1 + \cdots + \alpha_n$. This space has dimension $N(m, n) := (m+n)!/m!n!$, and the monomials ξ^α , $|\alpha| \leq m$, can be considered as a basis. On $\mathcal{P}(m)$ we consider the norm $\|\sum_{|\alpha| \leq m} c_\alpha \xi^\alpha\|_m := \sqrt{\sum_{|\alpha| \leq m} |c_\alpha|^2}$.

LEMMA 3.1 (see [3]). From m a finite set $A_m = \{\nu_i \mid 1 \leq i \leq L(m, n)\} \subseteq \mathbb{Q}^n$ of vectors, $L(m, n) := (m+1)^{(n+1)}$, can be computed such that for all $P \in \mathcal{P}(m)$, $P \neq 0$,

$$(8) \quad (\exists \nu \in A_m)(\forall z \in \mathbb{C}, |z| = 1)P(z\nu) \neq 0.$$

A_m can be chosen as follows: $A_m = \{\frac{k}{m}\nu : \nu \in A', k = 0, 1, \dots, m\}$, where $A' = \{\xi \in \mathbb{R}^n : \xi_i \in \{0, 1, 2, \dots, m\}, 1 \leq i \leq n\}$. Observe that $|A_m| = (m+1)^{(n+1)} = L(m, n)$.

LEMMA 3.2. There is a Type-2 Turing machine which, for every polynomial $P \in \mathcal{P}(m)$ of degree $m > 0$, computes a number $l \in \mathbb{N}$ such that

$$(9) \quad 2^{-l} < \max_{\nu \in A_m} \inf_{|z|=1} |P(\xi + z\nu)| \quad \forall \xi \in \mathbb{R}^n.$$

More precisely, from the degree m and ρ^2 -names of the coefficients $c_\alpha \in \mathbb{C}$ ($|\alpha| \leq m$) of a polynomial $P(\xi) = \sum_{|\alpha| \leq m} c_\alpha \xi^\alpha$ the machine computes a number l such that (9) holds.

Proof. By Lemma 3.1 there is some $\nu \in A_m$ such that for all $z \in \mathbb{C}$, $P(z\nu) \neq 0$ if $|z| = 1$. Since $z \mapsto |P(z\nu)|$ is continuous,

$$(10) \quad 0 < \max_{\nu \in A_m} \inf_{|z|=1} |P(z\nu)|.$$

For every $\nu \in A_m$ the total function

$$(\tilde{c}, z) \mapsto |P(z\nu)| \text{ is } (\rho^{2N(m,n)}, \rho^2, \rho)\text{-computable}$$

($\tilde{c} := (c_\alpha)_{|\alpha| \leq m} \in \mathbb{C}^{N(m,n)}$). By Lemma 2.1 on type conversion, the function

$$\tilde{c} \mapsto (z \mapsto |P(z\nu)|) \text{ is } (\rho^{2N(m,n)}, [\rho^2 \rightarrow \rho])\text{-computable.}$$

Since $\{z \in \mathbb{C} \mid |z| = 1\}$ is a κ^2 -computable compact set (see Definition 5.2.1 in [4]) the function

$$f \mapsto \inf_{|z|=1} f(z) \text{ is } ([\rho^2 \rightarrow \rho], \rho)\text{-computable}$$

by Corollary 6.2.5 in [4]. Therefore, for each $\nu \in A_m$, the function

$$(11) \quad \tilde{c} \mapsto \inf_{|z|=1} |P(z\nu)| \text{ is } (\rho^{2N(m,n)}, \rho)\text{-computable.}$$

By Theorem 6.2.1 in [4], the total function

$$(12) \quad \tilde{c} \mapsto \max_{\nu \in A_m} \inf_{|z|=1} |P(z\nu)| \text{ is } (\rho^{2N(m,n)}, \rho)\text{-computable.}$$

Since $B_m := \{\tilde{c} \mid \sqrt{\sum_{|\alpha| \leq m} |c_\alpha|^2} = 1\}$ is compact and $\tilde{c} \mapsto \max_{\nu \in A_m} \inf_{|z|=1} |P(z\nu)|$ is continuous by (12), by (10)

$$\inf_{\|\tilde{c}\|_m=1} \max_{\nu \in A_m} \inf_{|z|=1} |P(z\nu)| > 0.$$

Since B_m is $\kappa^{2N(m,n)}$ -computable, by (12) and Corollary 6.2.5 in [4],

$$\inf_{\|\tilde{c}\|_m=1} \max_{\nu \in A_m} \inf_{|z|=1} |P(z\nu)| > 0 \text{ is } \rho\text{-computable.}$$

Since all the above computations are uniform in the degree m , a rational number $C_m > 0$ can be computed from m such that

$$(13) \quad C_m < \max_{\nu \in A_m} \inf_{|z|=1} |P(z\nu)| \text{ if } \|P\|_m = 1.$$

Now consider arbitrary polynomials $P(\xi) = \sum_{|\alpha| \leq m} c_\alpha \xi^\alpha$ of degree m . There is a Type-2 machine which on input m and the coefficients $(c_\alpha)_{|\alpha| \leq m}$ computes some rational number a_P such that $0 < a_P < |c_\beta|$ for some index β with $|\beta| = m$.

For fixed $\xi \in \mathbb{R}^n$, $P(\xi + \xi') = \sum c_\alpha (\xi + \xi')^\alpha = \sum d_\alpha(\xi) (\xi')^\alpha$, where the coefficients of the polynomials $d_\alpha(\xi)$ can be determined by algebraic computation. By an easy observation, $d_\beta(\xi) = c_\beta$ since $|\beta| = m$. Let $Q(\xi') := P(\xi + \xi')$. Then $a_P < |c_\beta| \leq \sqrt{\sum |d_\alpha(\xi)|^2} = \|Q\|_m$. Since $Q/\|Q\|_m$ has norm 1, by (13), $C_m < \max_{\nu \in A_m} \inf_{|z|=1} |Q(z\nu)|/\|Q\|_m$, and therefore

$$a_P \cdot C_m < \|Q\|_m \cdot C_m \leq \max_{\nu \in A_m} \inf_{|z|=1} |Q(z\nu)| = \max_{\nu \in A_m} \inf_{|z|=1} |P(\xi + z\nu)|.$$

Finally, some $l \in \mathbb{N}$ can be computed such that $2^{-l} \leq a_P C_m$. \square

In the following we denote, for any $(\xi_1, \dots, \xi_n) \in \mathbb{R}^n$, $|(\xi_1, \dots, \xi_n)| := \max_j |\xi_j|$. \overline{X} will denote the closure, and X° will denote the interior of a set X . For vectors in \mathbb{R}^n let $(x_1, \dots, x_n) < (y_1, \dots, y_n)$ iff $x_i < y_i$ for all i . For $a, b \in \mathbb{R}^n$ and $a < b$, let $(a; b]$ be the half-open interval (box) $\{x \in \mathbb{R}^n \mid a < x \leq b\}$. Let $\mathcal{B}_1 := \{(a; b] \mid a, b \in \mathbb{Q}^n \text{ and } a < b\}$ be the set of all half-open rational intervals with canonical notation ν_{B_1} . The set $\{I^\circ \mid I \in \mathcal{B}_1\}$ is a basis of the topology on \mathbb{R}^n . Let \mathcal{B} be the set of all finite unions of elements from \mathcal{B}_1 , and let ν_B be a standard notation of \mathcal{B} . Notice that $\emptyset \in \mathcal{B}$ and \mathcal{B} is closed under union, intersection, and difference $(I, J) \mapsto I \setminus J$ and that every $J \in \mathcal{B}$ is a finite union of pairwise disjoint half-open intervals from \mathcal{B}_1 . Union and difference are (ν_B, ν_B, ν_B) -computable.

Integration of total continuous functions over intervals is computable [4]. We need a generalization to partial functions. The following lemma generalizes Theorem 6.4.1 in [4]. It can be proved by using standard techniques.

LEMMA 3.3.

1. The partial function $f \mapsto \int_{|z|=1} f(z) dz$ for continuous $f : \subseteq \mathbb{C} \rightarrow \mathbb{C}$, which is defined if $\{z \mid |z| = 1\} \subseteq \text{dom}(f)$, is $([\rho^2 \rightarrow_p \rho^2], \rho^2)$ -computable.
2. The partial function $(f, I) \mapsto \int_I f(\xi) d\xi$ for continuous $f : \subseteq \mathbb{R}^n \rightarrow \mathbb{C}$ and $I \in \mathcal{B}$, which is defined if $\overline{I} \subseteq \text{dom}(f)$, is $([\rho^n \rightarrow_p \rho^2], \nu_B, \rho^2)$ -computable.

Since the boundary of every $I \in \mathcal{B}$ has measure 0, for every continuous function f ,

$$\int_{\overline{I}} f(\xi) d\xi = \int_I f(\xi) d\xi = \int_{I^\circ} f(\xi) d\xi$$

if $\overline{I} \subseteq \text{dom}(f)$. After these preparations we can prove our main theorem.

THEOREM 3.4. There is a Type-2 Turing machine which for every differential operator $P(D) = \sum_{|\alpha| \leq m} c_\alpha D^\alpha \neq 0$ computes a fundamental solution $E \in \mathcal{D}'(\mathbb{R}^n)$. More precisely, from m and ρ^2 -names of the $c_\alpha \in \mathbb{C}$ it computes a $[\delta_D \rightarrow \rho^2]$ -name of a fundamental solution E .

Proof. First, we assume that the degree m of the polynomial $P(\xi) = \sum_{|\alpha| \leq m} c_\alpha \xi^\alpha$ is fixed. Let $\tilde{c} := (c_\alpha)_{|\alpha| \leq m} \in \mathbb{C}^{N(m,n)}$ be the vector of coefficients of $P(\xi)$. Let $A_m = \{\nu_i \mid 1 \leq i \leq L(m,n)\}$ be any set satisfying Lemma 3.1. Let $l \in \mathbb{N}$ be some constant such that (9). For $1 \leq j \leq L(m,n)$, define

$$(14) \quad \Omega_j := \left\{ \xi \in \mathbb{R}^n \mid 2^{-l} < \inf_{|z|=1} |P(\xi + z\nu_j)| \right\}.$$

Notice that $\bigcup_{j=1}^{L(m,n)} \Omega_j = \mathbb{R}^n$.

Let $M_{-1} := M_0 := \emptyset \in \mathcal{B}$ and $M_k := ((-k, \dots, -k); (k, \dots, k)) \in \mathcal{B}$ for $k \geq 1$. For $k, j \in \mathbb{N}$, $1 \leq j \leq L(m,n)$, let $T_j^k \in \mathcal{B}$ be sets such that

$$(15) \quad T_j^k \subseteq \Omega_j, \quad T_i^k \cap T_j^k = \emptyset \text{ for } i \neq j, \text{ and } \bigcup_j T_j^k = M_k \setminus M_{k-1}.$$

We shall show below how such sets T_j^k can be computed. Since for each k the T_j^k ($1 \leq j \leq L(m,n)$) are a partition of $M_k \setminus M_{k-1}$, and since the $M_k \setminus M_{k-1}$ are a partition of \mathbb{R}^n , $T_j^k \cap T_{j'}^{k'} = \emptyset$ if $k \neq k'$ or $j \neq j'$. Define $T_j := \bigcup_k T_j^k$. Then

$$(16) \quad T_j \subseteq \Omega_j, \quad T_i \cap T_j = \emptyset \text{ for } i \neq j, \text{ and } \bigcup_j T_j = \mathbb{R}^n.$$

For any $u \in \mathcal{D}(\mathbb{R}^n)$ and $k \in \mathbb{N}$ define

$$(17) \quad \tilde{E}_k(u) := (2\pi)^{-n/2} \sum_{j=1}^{L(m,n)} \int_{T_j^k} d\xi \frac{1}{2\pi i} \int_{|z|=1} \frac{\hat{u}(\xi + z\nu_j)}{P(\xi + z\nu_j)} \frac{1}{z} dz$$

and

$$(18) \quad \tilde{E}(u) := (2\pi)^{-n/2} \sum_{j=1}^{L(m,n)} \int_{T_j} d\xi \frac{1}{2\pi i} \int_{|z|=1} \frac{\hat{u}(\xi + z\nu_j)}{P(\xi + z\nu_j)} \frac{1}{z} dz.$$

The integrals exist because $u \in \mathcal{D}(\mathbb{R}^n)$ and $P(\xi + z\nu_j) \neq 0$ for $|z| = 1$ and $\xi \in T_j$ by (14) and (16).

Define E by $\langle E, \phi \rangle := \langle \tilde{E}, \mathcal{R}\phi \rangle$ for all $\phi \in \mathcal{D}(\mathbb{R}^n)$. Then $\langle \tilde{E}, \phi \rangle = \langle \tilde{E}, \mathcal{R}^2\phi \rangle = \langle E, \mathcal{R}\phi \rangle$ for all $\phi \in \mathcal{D}(\mathbb{R}^n)$. We show in the following that E is a fundamental solution of $P(D)$. By Lemma 2.2, E is a fundamental solution of $P(D)$ iff $\tilde{E}(P(D)v) = v(0)$ for any $v \in \mathcal{D}(\mathbb{R}^n)$. For any $v \in \mathcal{D}(\mathbb{R}^n)$ and $u = P(D)v$,

$$\begin{aligned} \tilde{E}(u) &= \tilde{E}(P(D)v) \\ &= (2\pi)^{-n/2} \sum_{j=1}^{L(m,n)} \int_{T_j} d\xi \frac{1}{2\pi i} \int_{|z|=1} \frac{\mathcal{F}(P(D)v)(\xi + z\nu_j)}{P(\xi + z\nu_j)} \frac{1}{z} dz \\ &= (2\pi)^{-n/2} \sum_{j=1}^{L(m,n)} \int_{T_j} d\xi \frac{1}{2\pi i} \int_{|z|=1} \frac{P(\xi + z\nu_j)\hat{v}(\xi + z\nu_j)}{P(\xi + z\nu_j)} \frac{1}{z} dz \\ &= (2\pi)^{-n/2} \sum_{j=1}^{L(m,n)} \int_{T_j} d\xi \frac{1}{2\pi i} \int_{|z|=1} \hat{v}(\xi + z\nu_j) \frac{1}{z} dz. \end{aligned}$$

We observe that

$$\begin{aligned} &\frac{1}{2\pi i} \int_{|z|=1} \hat{v}(\xi + z\nu_j) \frac{1}{z} dz \\ &= \frac{1}{2\pi i} \int_{|z|=1} (2\pi)^{-n/2} \int_{\mathbb{R}^n} e^{-ix(\xi+z\nu_j)} v(x) dx \frac{1}{z} dz \\ &= (2\pi)^{-n/2} \int_{\mathbb{R}^n} e^{-ix\xi} v(x) dx \frac{1}{2\pi i} \int_{|z|=1} \frac{e^{-ixz\nu_j}}{z} dz \\ &= (2\pi)^{-n/2} \int_{\mathbb{R}^n} e^{-ix\xi} v(x) e^{-ix0\nu_j} dx \quad (\text{Cauchy integral}) \\ &= (2\pi)^{-n/2} \int_{\mathbb{R}^n} e^{-ix\xi} v(x) dx \\ &= \hat{v}(\xi). \end{aligned}$$

Thus we obtain

$$\begin{aligned} \tilde{E}(P(D)v) &= (2\pi)^{-n/2} \sum_{j=1}^{L(m,n)} \int_{T_j} \hat{v}(\xi) d\xi \\ &= (2\pi)^{-n/2} \int_{\mathbb{R}^n} \hat{v}(\xi) d\xi \\ &= (2\pi)^{-n/2} \int_{\mathbb{R}^n} e^{i0\xi} \hat{v}(\xi) d\xi \\ &= v(0). \end{aligned}$$

Next we determine an upper bound of $|\tilde{E}_k(u)|$. Since $\hat{u} \in \mathcal{S}(\mathbb{R}^n)$,

$$(19) \quad (\exists k' \in \mathbb{N}) \quad (|\hat{u}(\xi)| |\xi|^{n+2} \leq 1 \quad \text{if} \quad |\xi| \geq k').$$

Define $\bar{\nu} := \max_{\nu \in A_m} |\nu|$.

Let $k \geq k' + \bar{\nu} + 1$ and $k \geq 2\bar{\nu} + 2$. Then for $\xi \in \mathbb{R}^n$, $z \in \mathbb{C}$, and $1 \leq j \leq L(m, n)$,

$$(20) \quad |\xi + z\nu_j| \geq |\xi| - \bar{\nu} \geq k' \quad \text{if} \quad |\xi| \geq k - 1 \quad \text{and} \quad |z| = 1.$$

We obtain

$$\begin{aligned} |\tilde{E}_k(u)| &\leq (2\pi)^{-n/2} \sum_{j=1}^{L(m,n)} \int_{T_j^k} d\xi \frac{1}{2\pi} \int_{|z|=1} \frac{|\hat{u}(\xi + z\nu_j)|}{|P(\xi + z\nu_j)|} \frac{1}{|z|} |dz| \\ &\leq 2^l (2\pi)^{-n/2} \sum_{j=1}^{L(m,n)} \int_{T_j^k} d\xi \frac{1}{2\pi} \int_{|z|=1} |\hat{u}(\xi + z\nu_j)| |dz| \quad (\text{by (14), (15)}) \\ &\leq 2^l (2\pi)^{-n/2} \sum_{j=1}^{L(m,n)} \int_{T_j^k} d\xi \frac{1}{2\pi} \int_{|z|=1} \frac{|\hat{u}(\xi + z\nu_j)| \cdot |\xi + z\nu_j|^{n+2}}{|\xi + z\nu_j|^{n+2}} |dz| \\ &\leq 2^l (2\pi)^{-n/2} \sum_{j=1}^{L(m,n)} \int_{T_j^k} d\xi \frac{1}{2\pi} \int_{|z|=1} \frac{1}{|\xi + z\nu_j|^{n+2}} |dz| \\ &\quad (\text{by (19) and (20), since } |\xi| \geq k - 1 \text{ for } \xi \in T_j^k) \\ &\leq 2^l (2\pi)^{-n/2} \sum_{j=1}^{L(m,n)} \int_{T_j^k} d\xi \frac{1}{2\pi} \int_{|z|=1} \frac{1}{(k - 1 - \bar{\nu})^{n+2}} |dz| \\ &\leq 2^l (2\pi)^{-n/2} \sum_{j=1}^{L(m,n)} \int_{T_j^k} d\xi \frac{1}{(k - 1 - \bar{\nu})^{n+2}} \\ &\leq 2^l (2\pi)^{-n/2} \frac{(2k)^n}{(k - 1 - \bar{\nu})^{n+2}} \quad (\text{by (15)}) \\ &\leq 2^l \cdot (2\pi)^{-n/2} \cdot 2^{2n+2} \frac{1}{k^2} \quad (\text{since } k \geq 2\bar{\nu} + 2). \end{aligned}$$

Since $\sum_{k>K} \frac{1}{k^2} \leq \frac{1}{K}$, we obtain for all $K \geq k' + 2\bar{\nu} + 2$,

$$\sum_{k=K+1}^{\infty} |\tilde{E}_k(u)| \leq 2^{l+2n+2} \cdot (2\pi)^{-n/2} \frac{1}{K},$$

and by (15)–(18),

$$(21) \quad \tilde{E}(u) = \sum_{k=0}^{\infty} \tilde{E}_k(u),$$

and for all $K \geq k' + 2\bar{\nu} + 2$,

$$(22) \quad \left| \tilde{E}(u) - \sum_{k=0}^K \tilde{E}_k(u) \right| \leq 2^{l+2n+2} \cdot (2\pi)^{-n/2} \frac{1}{K}.$$

It remains to prove that the fundamental solution E can be *computed* from the polynomial P . Concretely, a polynomial P of degree m is given by the list $\tilde{c} = (c_\alpha)_{|\alpha| \leq m}$, $c_\alpha \in \mathbb{C}$, of its coefficients represented by $\rho^{2N(m,n)}$. Let A_m be a set of vectors according to Lemma 3.1.

- (a) By Lemma 3.2, the multifunction $\tilde{c} \mapsto l$ such that (9) is $(\rho^{2N(m,n)}, \nu_{\mathbb{N}})$ -computable.
- (b) Next we compute the sets Ω_j defined in (14). For each number j the function $(l, \tilde{c}, \xi) \mapsto \inf_{|z|=1} |P(\xi + z\nu_j)| - 2^{-l}$ is $(\nu_{\mathbb{N}}, \rho^{2N(m,n)}, \rho^n, \rho)$ -computable. The proof is essentially that of (11). By Lemma 2.1 on type conversion, for each j the function

$$(l, \tilde{c}) \mapsto \left(\xi \mapsto \inf_{|z|=1} |P(\xi + z\nu_j)| - 2^{-l} \right)$$

is $(\nu_{\mathbb{N}}, \rho^{2N(m,n)}, [\rho^n \rightarrow \rho])$ -computable. For the open subsets of \mathbb{R}^n we use the representation $\theta_<$ defined by $\theta_<(p) = U$ iff $p \in \Sigma^\omega$ is (encodes) a list of all $I \in \mathcal{B}_1$ such that $\bar{I} \subseteq U$ (see Definition 5.1.15 in [4]). Notice that $U = \bigcup_{\bar{I} \subseteq U} I^\circ = \bigcup_{\bar{I} \subseteq U} \bar{I}$. Since the set $\{y \in \mathbb{R} \mid y > 0\}$ is reclusively enumerable (r.e.) open, by Theorem 6.2.4.1 in [4] for each j the function

$$(l, \tilde{c}) \mapsto \Omega_j \text{ is } (\nu_{\mathbb{N}}, \rho^{2N(m,n)}, \theta_<)\text{-computable.}$$

- (c) We describe how to compute sets T_j^k from sequences $p_j \in \Sigma^\omega$ such that $\theta_<(p_j) = \Omega_j$. Consider $k \in \mathbb{N}$. Simultaneously for all $j = 1, \dots, L(m, n)$ produce the lists of intervals $I \in \mathcal{B}_1$ encoded by the p_j . Halt as soon as a finite family $C \subseteq \mathcal{B}_1$ has been found such that $\overline{M_k \setminus M_{k-1}} \subseteq \bigcup_{I \in C} I^\circ$ (finite covering of a compact set). For each j let $C_j \subseteq C$ be the set of intervals from C so far listed by p_j . Let $T_0^k := \emptyset$ and determine the $T_j^k \subseteq \mathcal{B}$ successively by the rule

$$T_j^k := \left(\bigcup_{I \in C_j} I \cap (M_k \setminus M_{k-1}) \right) \setminus \bigcup_{j' < j} T_{j'}^k.$$

If (9) holds for the number l , then $\bigcup_j \Omega_j = \mathbb{R}^n$, a set C will be found by compactness of $\overline{M_k \setminus M_{k-1}}$, and (15) and (16) are true. As a summary, we have a multifunction

$$(l, \tilde{c}) \mapsto ((k, j) \mapsto T_j^k), \text{ which is } (\nu_{\mathbb{N}}, \rho^{2N(m,n)}, [\nu_{\mathbb{N}}^2 \rightarrow \nu_B])\text{-computable.}$$

- (d) Next we prove that $(\tilde{c}, k, u, ((k, j) \mapsto T_j^k)) \mapsto \tilde{E}_k(u)$ defined in (17) is $(\rho^{2N(m,n)}, \nu_{\mathbb{N}}, \delta_D, [\nu_{\mathbb{N}}^2 \rightarrow \nu_B], \rho^2)$ -computable. First let j be fixed. Since the identity from $\mathcal{D}(\mathbb{R}^n)$ to $\mathcal{S}(\mathbb{R}^n)$ is (δ_D, δ_S) -computable, the Fourier transform is (δ_S, δ_S) -computable [6], and the identity from $\mathcal{S}(\mathbb{R}^n)$ to $C(\mathbb{R}^n)$ is $(\delta_S, [\rho^n \rightarrow \rho^2])$ -computable, the partial function

$$(u, \tilde{c}, \xi, z) \mapsto \frac{\hat{u}(\xi + z\nu_j)}{P(\xi + z\nu_j)} \frac{1}{z} \text{ is } (\delta_D, \rho^{2N(m,n)}, \rho^n, \rho^2, \rho^2)\text{-computable.}$$

Therefore by type conversion (Lemma 2.1),

$$(u, \tilde{c}, \xi) \mapsto \left(z \mapsto \frac{\hat{u}(\xi + z\nu_j)}{P(\xi + z\nu_j)} \frac{1}{z} \right) \text{ is } (\delta_D, \rho^{2N(m,n)}, \rho^n, [\rho^2 \rightarrow_p \rho^2])\text{-computable.}$$

By Lemma 3.3 the function

$$(u, \tilde{c}, \xi) \mapsto \int_{|z|=1} \frac{\hat{u}(\xi + z\nu_j)}{P(\xi + z\nu_j)} \frac{1}{z} dz \quad \text{is } (\delta_D, \rho^{2N(m,n)}, \rho^n, \rho^2)\text{-computable.}$$

Again after type conversion by Lemma 3.3, the function

$$(I, u, \tilde{c}) \mapsto \int_I d\xi \int_{|z|=1} \frac{\hat{u}(\xi + z\nu_j)}{P(\xi + z\nu_j)} \frac{1}{z} dz \quad \text{is } (\nu_B, \delta_D, \rho^{2N(m,n)}, \rho^2)\text{-computable.}$$

Finally, since π is computable, we can conclude that

$$(\tilde{c}, k, u, ((k, j) \mapsto T_j^k)) \mapsto \tilde{E}_k(u)$$

is $(\rho^{2N(m,n)}, \nu_{\mathbb{N}}, \delta_D, [\nu_{\mathbb{N}}^2 \rightarrow \nu_B], \rho^2)$ -computable. By Theorem 3.1.7 in [4] on primitive recursion, the function

$$(\tilde{c}, K, u, ((k, j) \mapsto T_j^k)) \mapsto \sum_{k=0}^K \tilde{E}_k(u)$$

is $(\rho^{2N(m,n)}, \nu_{\mathbb{N}}, \delta_D, [\nu_{\mathbb{N}}^2 \rightarrow \nu_B], \rho^2)$ -computable.

- (e) As above, from $u \in \mathcal{D}(\mathbb{R}^n)$, $\hat{u} \in \mathcal{S}(\mathbb{R}^n)$ can be computed. By the definition of δ_S , a number k' can be computed from \hat{u} such that $|v(\xi)| |\xi|^{n+2} \leq 1$ if $|\xi| \geq k'$. From l, k' , and $L \in \mathbb{N}$, a number K can be computed such that $K \geq k' + 2\bar{\nu} + 2$ and $K \geq 2^{l+2n+2} \cdot (2\pi)^{-n/2} \cdot 2^{L+1}$. Then by (22),

$$\left| \tilde{E}(u) - \sum_{k=0}^K \tilde{E}_k(u) \right| \leq 2^{-L-1}.$$

By (d) above, from $\tilde{c}, K, u, ((k, j) \mapsto T_j^k)$ and L a rational $b_L \in \mathbb{C}$ can be computed such that

$$\left| b_L - \sum_{k=0}^K \tilde{E}_k(u) \right| \leq 2^{-L-1}.$$

From the sequence b_0, b_1, \dots we can compute a ρ^2 -name of $\tilde{E}(u)$. Therefore, after type conversion, the function

$$(\tilde{c}, l, ((k, j) \mapsto T_j^k)) \mapsto (u \mapsto \tilde{E}(u))$$

is $(\rho^{2N(m,n)}, \nu_{\mathbb{N}}, [\nu_{\mathbb{N}}^2 \rightarrow \nu_B], [\delta_D \rightarrow \rho^2])$ -computable.

From (a), (c), and (e), the multifunction $\tilde{c} \mapsto \tilde{E}$ is $(\rho^{2N(m,n)}, [\delta_D \rightarrow \rho^2])$ -computable. Since $\langle E, u \rangle = \langle \tilde{E}, \mathcal{R}u \rangle = \langle \mathcal{R}\tilde{E}, u \rangle$ and \mathcal{R} is computable on $\mathcal{D}'(\mathbb{R}^n)$ by Lemma 4.7 in [6], the operator $u \mapsto E$ mapping a polynomial of degree m to some fundamental solution is $(\rho^{2N(m,n)}, [\delta_D \rightarrow \rho^2])$ -computable.

So far we have assumed a fixed degree m of the polynomial. Since the set A_m can be computed from m (Lemma 3.1) and all the other computations are uniform also in m the proof is finished. \square

In the proof multivalued functions occur several times. A critical part is the computation of the sets T_j^k , which determines a partition of \mathbb{R}^n into (at most) $L(m, n)$

sets. The resulting distribution depends essentially on this partition. However, the function from $\mathbb{C}^{N(m,n)}$ to these sets cannot be computable (hence continuous) and single-valued at the same time, provided it is not trivial (Lemma 4.3.15 in [4]).

Problem. Is there a *single-valued* computable function mapping any polynomial of degree m to a fundamental solution?

As usual in recursion theory, we have not merely proved the *existence* of a computable function (from names of input objects to names of output objects), but also the proof is *constructive*; i.e., it explains how a concrete Turing machine or computer program can be constructed for computing this function. Of course, the algorithm is not trivial since subroutines for integration, for Fourier transform on Schwartz space, etc., have to be included. The presented algorithms can be improved, since, e.g., no derivatives of the Fourier transform \hat{u} of the input test function $u \in \mathcal{S}(\mathbb{R}^n)$ are needed for computing the $\tilde{E}_k(u)$ by integration. We finish the article with a question.

Problem. Can our informal algorithm be converted to a feasible numerical algorithm for computing fundamental solutions efficiently, or is the problem inherently difficult?

REFERENCES

- [1] J. BARROS-NETO, *An Introduction to the Theory of Distributions*, Pure Appl. Math. 14, Marcel Dekker, New York, 1973.
- [2] J. RAUCH, *Partial Differential Equations*, 2nd ed., Grad. Texts in Math. 128, Springer, New York, 1997.
- [3] F. TREVES, *Linear Partial Differential Equations with Constant Coefficients*, Gordon and Breach Science Publishers, New York, 1966.
- [4] K. WEIHRAUCH, *Computable Analysis*, Springer, Berlin, 2000.
- [5] K. WEIHRAUCH AND N. ZHONG, *An algorithm for computing fundamental solutions*, in *Computability and Complexity in Analysis*, Informatik Berichte FernUniversität in Hagen, vol. 320, V. Brattka, L. Staiger, and K. Weihrauch, eds., Proceedings of the Sixth International Workshop, CCA 2004, Lutherstadt Wittenberg, Germany, 2004, pp. 181–194.
- [6] N. ZHONG AND K. WEIHRAUCH, *Computability theory of generalized functions*, J. ACM, 50 (2003), pp. 469–505.

COMPACT LABELING SCHEME FOR ANCESTOR QUERIES*

SERGE ABITEBOUL[†], STEPHEN ALSTRUP[‡], HAIM KAPLAN[§],
TOVA MILO[§], AND THEIS RAUHE[‡]

Abstract. We consider the following problem. Given a rooted tree T , label the nodes of T in the most compact way such that, given the labels of two nodes u and v , one can determine in constant time, by looking only at the labels, whether u is ancestor of v . The best known labeling scheme is rather straightforward and uses labels of length at most $2\log_2 n$ bits each, where n is the number of nodes in the tree. Our main result in this paper is a labeling scheme with maximum label length $\log_2 n + O(\sqrt{\log n})$. Our motivation for studying this problem is enhancing the performance of web search engines. In the context of this application each indexed document is a tree, and the labels of all trees are maintained in main memory. Therefore even small improvements in the maximum label length are important.

Key words. ancestor queries, rooted tree, labeling algorithm, alphabetic codes

AMS subject classifications. 68W05, 05C78, 05C05

DOI. 10.1137/S0097539703437211

1. Introduction. The problem that we study in this paper is the following. Given a rooted tree T , what are the shortest labels that one can attach to the nodes of the tree such that, given the labels of two nodes v and w , it is possible to determine in constant time—by looking only at the labels—whether v is an ancestor of w ? When measuring the quality of a labeling scheme, we will look at the length of the maximal label, as a function of the size of the tree. We are interested in the *exact* function rather than its order of growth—constant factors do matter! (E.g., labels of length bounded by, say, $1.5 \log n$ bits will be considered significantly better than labels of length $2 \log n$ bits.¹) Note that clearly one cannot go below $\log n$ bits—if the maximal label has less than $\log n$ bits, there are not enough labels in the domain even to identify all nodes. But how much more is actually needed for answering ancestor queries? This somewhat abstract problem arises in optimizing web search engines.

Before presenting our results, let us consider a simple labeling scheme that we call the *DFS* labeling scheme. Let T be a tree of size n . To each internal node we assign a *first* and *last* value, and to a leaf we assign only a *first* value, as follows. Perform a depth first traversal [28] of the tree from the root r . Let $first(r) = counter = 0$. The first time a node v is visited, we set $first(v) = counter$ and increment the counter by 1. The last time an internal node v is visited, we set $last(v) = counter$ and increment the counter by 1. Now an internal node v is a proper ancestor of a node w if and only if $first(v) < first(w) < last(v)$. The standard binary representation of the numbers

*Received by the editors November 7, 2003; accepted for publication (in revised form) October 14, 2005; published electronically March 24, 2006. This paper includes the results from two conference papers [2, 7] presented at the 12th and 13th ACM-SIAM Symposia on Discrete Algorithms (SODA, 2001 and 2002).

<http://www.siam.org/journals/sicomp/35-6/43721.html>

[†]INRIA-Rocquencourt, 78153 Le Chesnay CEDEX, France (Serge.Abiteboul@inria.fr).

[‡]IT University in Copenhagen, Glentevej 67, DK-2400 Copenhagen NV, Denmark (stephen@it-c.dk, theis@it-c.dk).

[§]School of computer Science, Tel Aviv University, Tel Aviv 69978 (haimk@cs.tau.ac.il, milo@cs.tau.ac.il). The work of the third author was supported in part by Israel Science Foundation (ISF) grant 548/00.

¹All logarithms in this paper are base 2.

```

<Catalog>
  <Category>
    <Category_Name> Literature & Fiction </Category_Name>
    <Book>
      <ISBN>12445</ISBN> <Title>Bridget Jones's Diary</Title>
      <Authors><Author> Helen Fielding </Author></Authors>
      <Price> $10.39</Price> ...
    </Book>
    <Book> ... </Book>
    ...
  </Category>
  <Category> ... </Category>
  ...
</Catalog>

```

FIG. 1. *The books catalog.*

first and *last* consists of $\log n + 2$ bits. Hence, this scheme uses at most $\log n + 2$ bits for a label of a leaf and at most $2 \log n + 4$ bits for a label of an internal node. Interestingly, although rather naive, the bound obtained by this simple scheme is in fact the tightest we could find in the existing literature. We present a labeling scheme that improves this bound to $\log n + O(\sqrt{\log n})$ bits.

We use the RAM model of computation and assume that the length of a computer word is $\Omega(\log n)$ bits (hence the basic operations on the labels can be performed in constant time). The algorithm uses only the basic and fast RAM operations as assignment, less-than comparison, bitwise AND, OR, and XOR. Our labeling scheme avoids the sometime more costly operations such as multiplication.

Motivation. Most of the data on the web today is written in HTML format [33]. An HTML document consists of text interspersed with tag fields such as `<I>...</I>` to describe the presentation of the page, the inclusion of pictures, hyperlinks, forms, etc. These tags do not provide, however, any information about the semantic nature of the document components, which makes the analysis of the content of documents difficult. To remedy this, a new web standard, the XML data exchange format [31], has been proposed. Just like HTML, XML documents feature tags. But rather than providing instructions on how the document is to be displayed,² they provide information on the logical structure of the document, with each semantic element starting with an `<itemname>` tag and ending with an `</itemname>` tag. To see an example, consider Figure 1, which shows a fragment of an XML document describing a books catalog. The `<Category>` and `</Category>` tags are used to delimit the information corresponding to one catalog category. Each category item consists of a sequence of tagged subitems such as `Category_Name`, `Book`, etc.

XML is being widely adopted as web standard, and it is believed that although a large portion of the web will remain unstructured or hidden behind interactive graphics and forms, a large and contentwise essential portion of the web will soon be available in XML [25, 31].

To retrieve data from the web, people use search engines like Alta vista [8] or Google [17]. Today these search engines support full-text queries; namely, the user

²This may be specified in a *style-sheet*.

gives a few words, and the engine returns documents containing these particular words. When querying XML data it is desired also to support queries that utilize the document structure to ask for more specific data. For example, it is desired to support structural queries like “find all book items containing “Fielding” as an author and a price less than \$12” [1, 12, 32].

Going back to the problem stated in the introduction, observe that an XML document can be viewed as a tree (essentially the parse tree of the document) whose nodes are the document items and whose edges correspond to the component-of relationship among data items. With this view, a query of the above form amounts to finding nodes with particular tags (book, author, price) having certain ancestor relationship between them. In our example we ask for documents containing book nodes that are ancestors of particular qualifying author and price nodes.

To understand how to implement such structural queries, consider the following rather typical implementation of a search engine. The engine’s heart is a big hash table containing all words occurring in the database, where for each word we keep the identifiers of the documents in which it appears. For XML documents we refine this structure as follows. First, in addition to the words that appear in text items, we also keep the tags of the nodes (like “book,” “author,” etc.) in the hash table. Second, to allow structural queries we give each node in the XML document (tree) a unique label and associate with each tag in the hash table the labels of all nodes with this tag in each particular document containing it.

Now if we give the nodes meaningful labels that reflect the hierarchical structure of the documents (namely, such that, given labels of two items, we can determine whether one is a descendant, i.e., component, of the other), queries of the above form can be answered by using the index only, without access to the actual document. We first use the hash table to retrieve the labels of the relevant nodes, and then iterate over the retrieved label sets and select those having the appropriate ancestor relationships.

To allow for good performance it is essential that the index structure (or at least a large part of it) reside in main memory. Observe that we are talking here about an extremely large structure. Experiments to evaluate the effect of adding the structural information to the index show that adding a label of just one integer (4 bytes) to each word occurrence almost doubles the index size [34]. Since the length of the node labels is a main factor for the index size, reducing this, even by a constant factor, is a critical issue, contributing directly to hardware cost reduction and performance improvement.

Consider now the actual physical representation of the labels in the index. In a fixed-length representation each label is allocated a fixed space, determined by the maximal potential label length. In a variable-length representation labels are allocated a variable size space (consisting of a fixed prefix stating the actual size of the label, followed by the label itself), and the size of the index is determined by the average label length plus the log of the maximum potential length of a label (the space allocated for the fixed prefix). We study here the case of fixed-length representation, aiming to improve the bound on the maximal label length. The case of variable-length representation is left for future research.

Related work. Variants of the *DFS* labeling scheme have been described in several papers including [11, 18, 23, 27, 30]. Implementations of such approaches are integrated in the Xyleme XML warehouse system [34]. As explained by Kannan, Naor, and Rudich [18], the names of the nodes in a graph³ typically convey no in-

³Typically the integers 1 to n .

formation about the graph itself, and so memory is wasted. Therefore [18] focus on labeling the nodes in such a way that various information can be, as in this paper, detected solely by looking at names of the nodes. Kannan, Naor, and Rudich [18] considered determining ancestor relationship, and to that end they suggested a variant of the *DFS* labeling scheme. They also considered the parent relation and other information.

Extracting local information from the node is helpful for some routing applications. In [29] Thorup and Zwick show how to assign short labels to a packet which is to be routed, such that one can determine the ancestor relationship between the packet's destination and the currently visited node, thus avoiding costly access to external memory. Specifically, after the work presented in [7], Thorup and Zwick [29], independently, obtained a labeling scheme for ancestor queries with labels of length $\log n + O(\log n / \log \log n)$. Recently, Alstrup, Bille, and Rauhe [3] established that labels which carry ancestor information for every tree must have length $\log n + \Omega(\log \log n)$ for some trees. An experimental comparison of different labeling schemes for testing ancestor relationships on real XML data can be found in [20].

Labeling schemes for other types of queries have recently received significant interest. Labeling schemes for various distance queries have been studied in [15, 19, 22, 26]. Alstrup et al. [4] studied a labeling scheme for computing nearest common ancestors in trees. Labels for vertex-connectivity and the adjacency relation are described in [9, 10, 18, 19, 21, 27]. In [14] one finds an extensive survey of labeling schemes and their applications.

The rest of this paper is organized as follows. Section 2 gives some definitions and background on labelings of trees, alphabetic codes, and partitions of trees. We present our labeling scheme in section 3. In section 4 we describe the algorithm that answers a query based on two labels. Section 5 analyzes the lengths of the labels. Finally, in section 6 we show how to implement our algorithm so that it runs in linear time.

2. Preliminaries. In this section we give a more formal definition of labeling scheme. Furthermore, we define an alphabetic code and subtree partition of a tree, and state the known results we use to construct our labeling.

Labeling schemes. An ancestor labeling scheme for a family \mathcal{F} of trees with n nodes consist of two mappings EC and DC . For a given tree $T \in \mathcal{F}$, the encoder EC labels the nodes of the tree; i.e., $EC_T = EC(T)$ maps the nodes from T into labels. The decoder DC maps a pair of labels into $\{0, 1\}$ such that for all T , $DC(EC_T(v), EC_T(w)) = 1$ if and only if v is an ancestor to w in T . Notice that DC is independent of the tree from which the pair of labels is taken; it depends only on the two labels. In different papers [18, 21] there are additional requirements, such as different time constraints on the function DC . The function DC presented in this paper can be computed in constant time on a RAM, assuming a standard set of operations as explained in the introduction.

Alphabetic codes. We denote by $\langle y \rangle_k$ a sequence of objects y_1, y_2, \dots, y_k (such as integers or binary strings). For binary strings $a, b \in \{0, 1\}^*$, $a <_{\text{lex}} b$ if and only if a precedes b in the lexicographic order on binary strings, i.e., a is prefix of b , or the first bit in which a and b differ is 0 in a and 1 in b . A sequence $\langle b \rangle_k$ of binary strings is *lexicographically ordered* if $b_i <_{\text{lex}} b_j$ for all $1 \leq i < j \leq k$. Let $|s|$ denote the length of a binary string $s \in \{0, 1\}^*$. Observe that, given machine words that contain $a, b, |a|$, and $|b|$ in their least significant bits, it is possible to determine whether $a <_{\text{lex}} b$ in a constant number of operations as follows. Assume $|a| \leq |b|$. We first align a and b by

shifting a to the left by $|b| - |a|$ bits. Then we use standard integer comparison on the resulting words and return the result of this integer comparison if these words differ. If the two words are equal—this will happen if b equals a followed by a sequence of zeros—we break the tie according to the lengths of a and b .

For a sequence of binary strings $\langle b \rangle_k$, we say it is *prefix-free* if no string is a prefix of another in the sequence. The following lemma due to Gilbert and Moore [16] states the existence of a lexicographically ordered prefix-free sequence of binary strings called an alphabetic code.

LEMMA 2.1 (see Gilbert and Moore [16]). *For any sequence $\langle y \rangle_k$ of positive integers with $n = \sum_{i=1}^k y_i$ there exists a prefix-free lexicographically ordered sequence $\langle b \rangle_k$, where $|b_i| \leq \log n - \log y_i + O(1)$ for all i .*

For the sake of completeness we show how to construct an alphabetic code for an integer sequence $\langle y \rangle_k$ in $O(k)$ time. Let $s_0 = 0$, $s_i = \sum_{j=1}^i y_j$ for $1 \leq i \leq k$, $I_i = [s_{i-1} + 1, s_{i-1} + y_i]$, and $f_i = \max\{0, \lfloor \log y_i \rfloor\}$. For each y_i we will find a number z_i in the interval I_i such that z_i can be represented in a word with $w = \lceil \log n \rceil$ bits, having the f_i less significant bits set to 0. Then we can let b_i be the bit string consisting of the $w - f_i$ most significant bits of z_i . Thus we get alphabetic codes where $1 \leq |b_i| \leq \log n - \log y_i + O(1)$. We choose z_i such that $z_i + 2^{f_i} - 1$ belongs to I_i to make the bit strings prefix-free. The computation can be done as follows. In the interval I_i there must be a number z_i such that $z_i \bmod 2^{f_i} = 0$. If $s_{i-1} + 1 \bmod 2^{f_i} = 0$, then $z_i = s_{i-1} + 1$. Otherwise $z_i = s_{i-1} + 1 - (s_{i-1} + 1 \bmod 2^{f_i}) + 2^{f_i}$.

The algorithm runs in $O(k)$ time if machine operations for shift, remainder in a division by a power of 2, and discrete logarithm on words of $O(\log n)$ bits are available. In a machine that does not support such operations we can construct a table representing these functions, using $O(n)$ time and space. This will only increase the preprocessing time of our labeling algorithm by a constant factor. Note that Mehlhorn [24] gives a somewhat more complicated algorithm that produces an *alphabetic code*, $\langle b \rangle_k$, for an arbitrary sequence $\langle y \rangle_k$ of positive real numbers with the same bound on the lengths. Mehlhorn’s algorithm can be implemented to run in $O(k)$ time.

Tree terminology. We denote the sets of nodes and edges in T by $V(T)$ and $E(T)$, respectively. We let $T(u)$ denote the subtree of T rooted at node $u \in V(T)$. If $w \in V(T(u))$, then u is an *ancestor* of w , and we write $u \prec w$. If $w \in V(T(u)) \setminus \{u\}$, then u is a *proper ancestor* of w . If u is a (proper) ancestor of w , then w is a (*proper*) *descendant* of u . For nodes u and v in T we denote the path between u and v including u and v by $u \rightsquigarrow v$.

We say that a rooted tree is a *binary tree* if all nodes in the tree have at most two children. If a tree T is not binary, it is straightforward to construct a binary tree T' such that $|V(T')| \leq 2|V(T)|$, and a mapping $f : V(T) \rightarrow V(T')$ such that for $v, w \in V(T)$ we have that $v \prec_T w$ if and only if $f(v) \prec_{T'} f(w)$. Hence, in the remainder of this paper we assume without loss of generality that T is a binary tree.

Clustering. Let T be a rooted tree of size $n = |V(T)| > 1$. Let C be a connected subgraph of T . A node x in $V(C)$ is a *boundary node* if either $x = r$, where r is the root of T , or x is adjacent to a node in $V(T) \setminus V(C)$. The boundary nodes of C are denoted by ∂C . A *cluster* is a connected subgraph of T , where $|\partial C| \leq 2$. Clusters are also used in [5, 6, 13].

Next we define a *cluster partition* of a tree T . If T is a single vertex, then its *cluster partition* consists of one cluster containing this vertex. Otherwise a set of

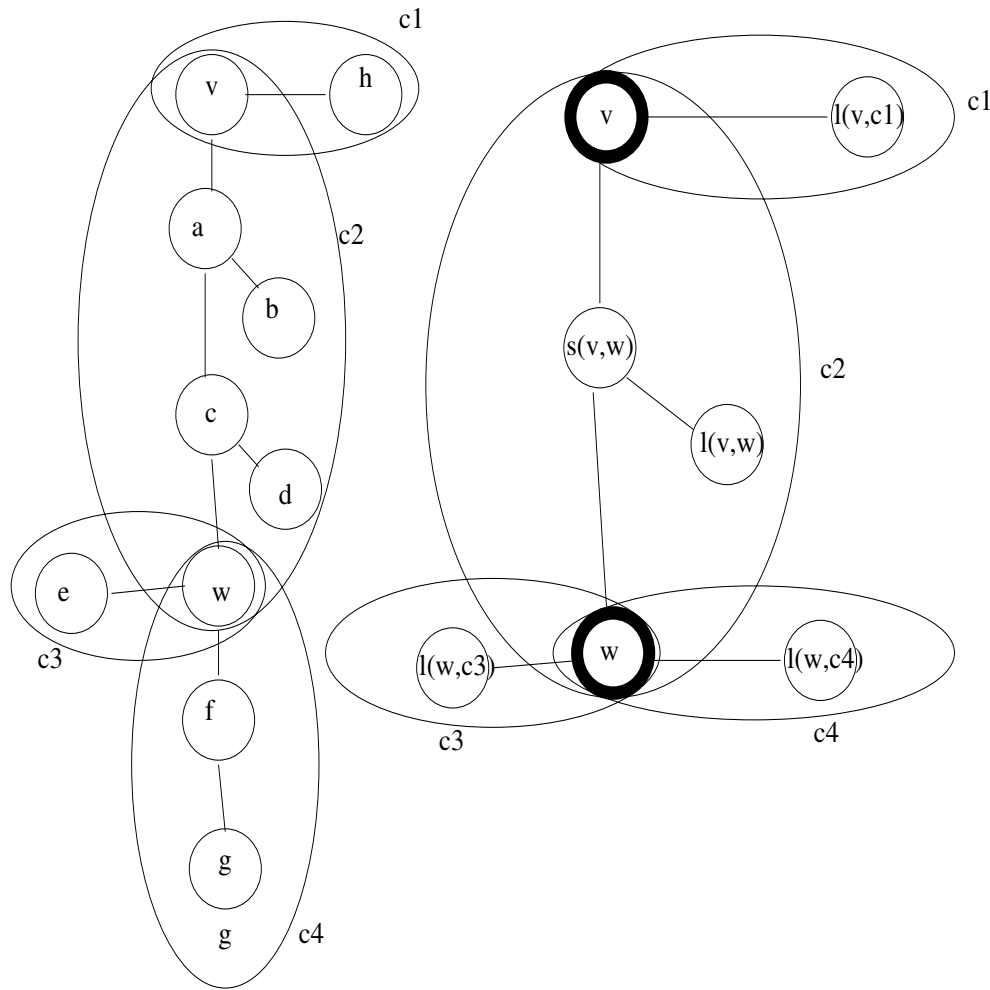


FIG. 2. *Left: a tree T decomposed into four clusters, where $V(c1) = \{v, h\}$, $\partial c1 = \{v\}$, $V(c2) = \{v, a, b, c, d, w\}$, $\partial c2 = \{v, w\}$, $V(c3) = \{w, e\}$, $\partial c3 = \{w\}$, $V(c4) = \{w, f, g\}$, and $\partial c4 = \{w\}$. Right: the tree T^m , illustrating from which clusters the nodes of T^m originate.*

clusters CS is a *cluster partition* of a tree T if the following conditions hold:

1. For every $C \in CS$, $|E(C)| \geq 1$.
2. $V(T) = \cup_{C \in CS} V(C)$, and $E(T) = \cup_{C \in CS} E(C)$.
3. For any $C_1, C_2 \in CS$, $E(C_1) \cap E(C_2) = \emptyset$.
4. If $v \in \partial C$ has two children a and b , then no cluster includes both a and b .

Note that the sets of edges of the different clusters form a partition of the set of edges of T . A vertex, on the other hand, may belong to more than one cluster. Figure 2 gives an example of a cluster partition. A variant of the following lemma was also used in [5, 6, 13].

LEMMA 2.2. *Given a tree T , $n > 1$, and a parameter x , where $\lceil n/x \rceil \geq 2$, it is possible to construct a cluster partition CS in linear time such that $|CS| \leq k \cdot x$ for a constant k , and $|V(C)| \leq \lceil n/x \rceil$ for every $C \in CS$.*

Proof. We describe an algorithm that constructs such a cluster partition. Let S be the set of boundary nodes picked until now; initially $S = \{r\}$. Now recursively

choose a node $v \in S$ not yet examined. Let the children of v be a and b . We treat the two children similarly; hence in the following we can assume $c = a$ (or $c = b$). If $|V(T(c))| + 1 \leq \lceil n/x \rceil$, we let the nodes $V(T(c))$ and v be a cluster, with the boundary node v . Otherwise, let $w \in V(T(c))$ be a maximal node such that $|V(T(c))| + 2 - |V(T(w))| \leq \lceil n/x \rceil$. (By *maximal* we mean that this inequality holds for w but does not hold for any of the children of w .) We let $V(T(c)) \setminus V(T(w)) \cup \{v, w\}$ be a cluster, with boundary nodes w and v , and add w to S . Finally, we remove v from S . It is easy to see that one can implement this algorithm to run in linear time.

Clearly we have $|V(C)| \leq \lceil n/x \rceil$ for every $C \in CS$. Notice that each node is a boundary node in at most three different clusters, and each cluster has at least one boundary node. Therefore the number of boundary nodes is within a constant factor of the number clusters. So to bound the number of clusters we will prove an upper bound on the number of boundary nodes.

To bound the number of boundary nodes, we consider separately those boundary nodes that have two children that are roots of subtrees of T of size $\geq \lceil n/x \rceil$, and those boundary nodes which have at least one child which roots a subtree of size $\leq \lceil n/x \rceil - 1$.

Consider first the set S of boundary nodes which have two children that are roots of subtrees of size $\geq \lceil n/x \rceil$. It is easy to see that, by the definition of our algorithm, if y and z are nodes in S , then their lowest common ancestor must be a boundary node and therefore belongs to S . Thus if we connect each node in S to its immediate descendants that are in S , we obtain a tree with all internal nodes having two children. Since the number of leaves in this tree is $O(x)$ (each roots a tree with at least $2\lceil n/x \rceil + 1$ nodes), we obtain that there are $O(x)$ boundary nodes in S .

Consider now a boundary node z that has one child y which roots a subtree of size $\leq \lceil n/x \rceil - 1$. By the definition of our algorithm, z and the subtree rooted by y form a cluster C . Assume that z is not the root. Then by the definition of our algorithm, the cluster C , and the other one or two clusters for which z is a boundary node, contain together at least $\lceil n/x \rceil$ nodes. (Otherwise z should not have been selected as a boundary node.) Therefore, if we charge each such boundary node evenly among the nodes of the clusters of which it is a boundary, each node in these clusters is charged by at most x/n . Since each node is charged by a constant number of boundary nodes, the total charge of a node is $O(x/n)$, and the total charge summed over all nodes is $O(x)$. Thus there are $O(x)$ boundary nodes with at most one child which roots a subtree of size $\leq \lceil n/x \rceil - 1$. \square

3. A $\log n + O(\sqrt{\log n})$ labeling scheme. Our labeling scheme is based on a particular tree decomposition technique. We define this tree decomposition in section 3.1 and characterize the ancestor relationship in terms of it. The labeling itself is defined in section 3.2.

3.1. Tree decomposition. Let T be a binary tree with root r . (As explained in section 2 when we defined our tree terminology, we assume that T is a binary tree.) Let CS be a cluster partition of T , as described in Lemma 2.2. We will fix the parameter x later. By the definition of a cluster partition we have that, if z is a boundary node in a cluster, it is a boundary node in every cluster that contains it. If z not is a boundary node, it belongs to a unique cluster, which we denote by $C(z)$. Furthermore, for $v, w \in \partial C$, either v or w is an ancestor in T of all nodes in $V(C)$. Let C be a cluster such that $\partial C = \{v, w\}$; we define the *spine path* of C to be the set of nodes on the path from v to w in T excluding v and w . We denote this path by $\mathcal{P}(v, w)$. In the example of Figure 2, $\mathcal{P}(v, w) = \{a, c\}$.

Given a cluster partition CS of T , we define a *macro tree* and denote it by T^m as follows. The vertex set of T^m consists of the boundary nodes of all clusters in CS , together with one or two new nodes for each cluster. For a cluster C with a single boundary node, say $\partial C = \{v\}$, we introduce one new node, denoted by $\ell(v, C)$. For a cluster C with two boundary nodes, say $\partial C = \{v, w\}$, where v is an ancestor of w , we introduce two new nodes denoted by $\ell(v, w)$ and $s(v, w)$. The edges of T^m consist of an edge $(v, \ell(v, C))$, for every cluster C with a single boundary node, and the three edges $(v, s(v, w))$, $(s(v, w), w)$, and $(s(v, w), \ell(v, w))$, for every cluster C with two boundary nodes. See Figure 2.

Since r is a boundary node, then $r \in V(T^m)$. Furthermore, since there is a path from r to every boundary node $v \in T$, there is a corresponding path from r to v in T^m , and therefore T^m is connected. We can obtain T^m by deleting the edges that do not belong to spine paths and then replacing each spine path $\mathcal{P}(v, w)$ by a path of two edges connecting v and w through $s(v, w)$. Finally for every cluster with a single boundary node we add a leaf $\ell(v, C)$ and connect it to v , and we also connect a leaf $\ell(v, w)$ to every node $s(v, w)$. From this description it is clear that T^m is indeed a tree. We define r to be the root of T^m . The properties of T^m that are summarized in the following lemma easily follow from Lemma 2.2 and the definition of T^m .

LEMMA 3.1. *Let T be a binary tree, and let T^m be the macro tree corresponding to a cluster partition of T . Then*

- T^m is a binary tree;
- $|V(T^m)| \leq 4kx$, where k is the constant given in Lemma 2.2;
- the leaves of T^m are the nodes $\ell(\cdot, \cdot)$;
- for two boundary nodes $v, w \in T$, $v \prec_T w$ if and only if $v \prec_{T^m} w$.

We classify each node of $v \in T$ into one of four different types as follows:

1. *Boundary node:* v is a boundary node of a cluster in the cluster partition.
2. *Spine node:* $C(v)$ has two boundary nodes, and v is on the spine path of $C(v)$.
3. *Leaf clustered node:* $C(v)$ has a single boundary node, and v is in $C(v) \setminus \partial C(v)$.
4. *Internal clustered node:* $C(v)$ has two boundary nodes, say u and w , and v is in $C(v) \setminus \{u, w\} \setminus \mathcal{P}(u, w)$.

With every node $v \in T$ we associate a *representative node* in T^m , which we denote by $a(v)$, as follows. If v is a boundary node, then $a(v) = v$. If v is a spine node, where $\partial C(v) = \{u, w\}$, then $a(v) = s(u, w)$. If v is a leaf clustered node, then $a(v) = \ell(v, C(v))$, and if v is an internal clustered node, then $a(v) = \ell(u, w)$, where u and w are the boundary nodes of $C(v)$. In the example of Figure 2, v and w are boundary nodes, a and c are spine nodes, b and d are internal clustered nodes, and e , f , g , and h are leaf clustered nodes.

Let $z \in T$ be a spine node. By the definition of a spine node, node z has a child w , which is either a boundary node or a spine node. We define $Isubtree(z)$ to be $T(z)$ from which we have removed the subtree $T(w)$ and the edge (w, z) . It follows that $V(Isubtree(z))$ consists of z and all the internal clustered nodes in $C(z)$ whose nearest node on the spine path is z . For $w \in V(Isubtree(z))$, we define $sub(w) = z$. Thus, the $sub()$ mapping maps each internal clustered node to its closest ancestor on the spine path of its cluster. For nodes v and w on the same spine path, we define an ordering $<_\alpha$ such that $v <_\alpha w$ if and only if $depth_T(v) < depth_T(w)$.

It would be instructive at this point to understand how the ancestor relationship between nodes in T relates to the classification of the nodes and their representatives in T^m . We summarize this relation in the following lemma.

LEMMA 3.2. *Letting $v \in T$, the followings cases give necessary and sufficient conditions for $v \prec_T w$ to hold, according to the type of v :*

1. *If v is a leaf clustered node, then $v \prec w$ if and only if w is also a leaf clustered node, $a(v) = a(w)$, and v is an ancestor of w in $C(v)$.*
2. *If v is an internal clustered node, then $v \prec w$ if and only if w is also an internal clustered node, $a(v) = a(w)$, $sub(v) = sub(w)$, and $v \prec_{Isubtree(sub(v))} w$.*
3. *If v is a spine node, then $v \prec w$ if and only if one of the following conditions holds:*
 - (a) $C(v) = C(w)$, w is also a spine node, and $v <_\alpha w$.
 - (b) $C(v) = C(w)$, w is an internal clustered node, and $v = sub(w)$ or $v <_\alpha sub(w)$.
 - (c) $C(v) \neq C(w)$ and $a(v)$ is an ancestor of $a(w)$ in T^m .
4. *If v is a boundary node, then $v \prec w$ if and only if $a(v)$ is an ancestor of $a(w)$ in T^m .*

Lemma 3.2 indicates that we can compute ancestor relationship in T by computing one of the following:

1. ancestor relationship in T^m ,
2. ancestor relationship in $Isubtree(v)$ for a spine node v ,
3. ancestor relationship in a cluster C with a single boundary node,
4. whether $v <_\alpha w$ for two spine nodes v and w .

We achieve 1 by using a DFS labeling of T^m . We achieve 2 and 3 by recursively decomposing clusters with a single boundary node and subtrees hanging off spine nodes. We achieve 4 by using an alphabetic code for the spine nodes on each spine path. The details of how exactly we construct the labels are in the next section.

3.2. The labels. Let $L^m : V(T^m) \rightarrow \{0, 1\}^*$ be the DFS labeling of T^m . Recall that this DFS labeling associates a label consisting of $|L^m(v)| \leq 2 \log x + O(1)$ bits with each internal node v in T^m , and a label of $|L^m(\ell)| \leq \log x + O(1)$ bits with each leaf ℓ in T^m . For binary strings b_1 and b_2 we denote the concatenation of b_1 and b_2 by $b_1 \cdot b_2$.

The label of each node starts with a prefix of constant length stating the type of the node. We denote the prefix corresponding to a boundary node, a spine node, an internal clustered node, and a leaf clustered node, by `boundary`, `spine`, `int_clustered`, and `leaf_clustered`, respectively. The bits following this constant length prefix depend on the type of the node v as follows.

Boundary node. In addition to the type bits, $L(v)$ contains the label of v in T^m .

That is,

$$L(v) = \text{boundary} \cdot L^m(a(v)).$$

It follows that in this case $|L(v)| \leq 2 \log x + O(1)$.

Spine node. Let $a(v) = s(v', w') \in T^m$, and let $SP = \mathcal{P}(v', w')$ be the spine path of $C(v)$. Let $k = |SP|$ be the number of nodes on this spine path, and let v_1, v_2, \dots, v_k be the nodes on SP ordered such that for $i < i'$, $depth_T(v_i) < depth_T(v_{i'})$. Clearly v is a node on this spine path, say $v = v_j$, for some $1 \leq j \leq k$. Let $T_i = Isubtree(v_i)$, and let $\langle c \rangle_k$ be an alphabetic code of the sequence $\langle |V(T_1)|, \dots, |V(T_k)| \rangle$ constructed according to Lemma 2.1. Hence, c_j is the string corresponding to $V(T_j)$ in $\langle c \rangle_k$. Note that $|c_j| \leq \log \sum_i |V(T_i)| - \log |V(T_j)| + O(1) \leq \log(n/x) - \log |V(T_j)| + O(1)$, since all nodes in T_i , $1 \leq i \leq k$, belong to the same cluster which contains no more

than n/x nodes. The label of v is defined to be

$$L(v) = \mathbf{spine} \cdot L^m(s(v', w')) \cdot c_j.$$

Thus, $|L(v)| \leq 2 \log x + \log(n/x) - \log |V(T_j)| + O(1) \leq \log x + \log n + O(1)$.
Internal clustered node. Let $a(v) = \ell(v', w')$, and let $v_j = \mathit{sub}(v)$ be the spine node associated with v . As in the previous case, let v_1, \dots, v_k be the nodes on the spine path $\mathcal{P}(v', w')$, and let $\langle c \rangle_k$ be an alphabetic code for the sequence $\langle |V(T_1)|, \dots, |V(T_k)| \rangle$, where $T_i = \mathit{Isubtree}(v_i)$. Thus c_j is the string corresponding to T_j in the code $\langle c \rangle_k$. By its definition the root of T_j has one child d in T_j . We define $T'_j = T_j(d)$ to be the subtree of T_j rooted by this child. We recursively apply the labeling algorithm to T'_j (with the same decomposition threshold x , which is not scaled relative to the size of T'_j). Let $L^b(v)$ denote the label of node v in the recursive labeling of T'_j . The label of v in the tree T is defined to be

$$L(v) = \mathbf{int_clustered} \cdot L^m(\ell(v', w')) \cdot c_j \cdot L^b(v).$$

Thus, $|L(v)| \leq \log x + \log(n/x) - \log |T_j| + |L^b(v)| + O(1) = \log n - \log |T_j| + |L^b(v)| + O(1)$, where $|L^b(v)|$ is the length of the recursive ancestor labeling of v in T'_j .

Leaf clustered node. Let c be the only child of the boundary node of $C(v)$ in $C(v)$. We recursively label the tree $T(c)$ (with the same decomposition threshold x , which is not scaled relative to the size of $T(c)$). Let $L^b(v)$ be the label of v in this labeling of $T(c)$. Also let $a(v) = \ell(v', C)$. The label of v is defined to be

$$L(v) = \mathbf{leaf_clustered} \cdot L^m(\ell(v', C)) \cdot L^b(v).$$

We obtain in this case that $|L(v)| \leq \log x + |L^b(v)| + O(1)$, where $|L^b(v)|$ is the length of $L^b(v)$.

We say that the prefix of $L(v)$ before the recursive labeling $L^b(v)$ is a *block* of the labeling $L(v)$. Similarly $L^b(v)$ consists of such recursive blocks, i.e., $L(v) = b_1 b_2, \dots, b_k$, where $b_i \in \{0, 1\}^*$ is a block of bits starting with the type bit information and ending just before the type bit information of block b_{i+1} . By the definition of the labels, the last block b_k is either a label of a spine node or a label of a boundary node.

The recursion ends when we get to label a tree of size no larger than x , in which case $\lceil |T|/x \rceil \leq 1$ and Lemma 2.2 does not apply.

In this case we consider all nodes as boundary nodes and take the macro tree T^m to be identical to T .

Consider a node $v \in V(T)$ and its label $L(v)$ with blocks b_1, \dots, b_k . We define subtrees of T relative to prefixes of this label. For the empty bitstring ϵ we let $T(\epsilon) = T$. The tree $T(b_1)$ is the subtree of T containing v , whose recursive labeling defined the suffix of the label of v to be b_2, \dots, b_k . In general for every $i < k$ we let $T(b_1 b_2, \dots, b_i)$ denote the subtree of T containing v whose recursive labeling defined the suffix of the label of v to be b_{i+1}, \dots, b_k .

In addition to the labeling defined above, we also keep the index of the first bit of the last block b_k , i.e., $|b_1, \dots, b_{k-1}|$, using $\log \log n$ bits. This index, which is of fixed length, is stored at a fixed position of the word storing the label. Note that each block b_i stores a DFS label of a node in the macro tree corresponding to $T(b_1 b_2, \dots, b_{i-1})$. This DFS label occupies a fixed number of bits at the beginning of the block. This

number of bits is either $2 \log x + O(1)$ in case of a boundary node or a spine node, or $\log x + O(1)$ in case of an internal clustered node or a leaf clustered node.

To parse the labels while answering a query, the decoder needs to know the length and the position of the field storing $|b_1, \dots, b_{k-1}|$. Furthermore, it also needs to know our decomposition threshold x . The decoder has this information if it knows n , the size of the tree, since both these lengths are functions of n : the first is $\log \log n$ and the second will be fixed to $2^{\sqrt{\log n}}$. Since both lengths change quite slowly with n , the decoder does not in fact have to know n exactly but just up to a precision that allows it to determine these values uniquely. For example, knowing $\log n$ certainly suffices. In case the value of $\log n$ is not available to the decoder, one can encode it within each label using $\log \log n$ bits.

4. Answering a query. Let v, w be two nodes in the tree T with labels $L(v) = a_1 a_2, \dots$ and $L(w) = b_1 b_2, \dots$, where a_i and b_i are the blocks of the labels of v and w , respectively, as defined in section 3.2. We now show how to decide based on $L(v)$ and $L(w)$ whether $v \prec_T w$. The test consists of two phases defined as follows.

Let j be the index of the last block of $L(v)$, and assume first that $j > 1$. (If $j = 1$, we do nothing in the first phase and jump directly to the second phase.) It follows from Lemma 3.2 that if $v \prec w$, then v and w must both be either leaf clustered nodes or internal clustered nodes in the same cluster of the cluster partition of $T_i = T(a_1, a_2, \dots, a_i)$ for every $i < j$. Furthermore, in each of these cluster partitions where they are internal clustered nodes then it must be the case that $sub(w) = sub(v)$. Therefore, if $v \prec w$ and $j > 1$, we must have that for every $i < j$, $a_i = b_i$. Thus in the first phase of the query algorithm we verify that for every $i < j$, $a_i = b_i$. If this test fails, we report that $v \not\prec_T w$, and otherwise we continue to the second phase.

To efficiently test that indeed $a_i = b_i$ for every $i < j$ without decomposing $L(v)$ and $L(w)$ into their components a_1, a_2, \dots, a_{j-1} and b_1, b_2, \dots, b_{j-1} , respectively, we use the fact that a_i cannot be a prefix of b_i and b_i cannot be a prefix of a_i . This follows from the fact that the part of a_i and b_i containing a DFS label of a node in $T_{i-1} = T(a_1, a_2, \dots, a_{i-1})$ is of fixed length, and in case v and w are internal clustered nodes of the same cluster in T_{i-1} also from the fact that the strings in an alphabetic code are prefix-free.

Since a_i is not a prefix of b_i and b_i is not a prefix of a_i , one can easily prove by induction that $a_1 a_2, \dots, a_{j-1}$ is a prefix of $b_1 b_2, \dots$ if and only if $a_i = b_i$ for every $1 \leq i \leq j - 1$. Thus we start the processing of the query by testing if $a_1 a_2, \dots, a_{j-1}$ is a prefix of $L(w)$. Note that we can extract $a_1 a_2, \dots, a_{j-1}$ from $L(v)$ since we store with $L(v)$ the index of the first bit of b_j . If $a_1 a_2, \dots, a_{j-1}$ is not a prefix of $L(w)$, then v is not an ancestor of w . Otherwise we proceed to the second phase of the algorithm, which is defined as follows.

Let $\bar{T} = T(a_1, a_2, \dots, a_{j-1})$. Next we check whether $v \prec w$ in \bar{T} since at this point we know that $v \prec w$ in T if and only if $v \prec w$ in \bar{T} . Since j is the index of the last block of $L(v)$, node v is either a spine node or a boundary node in the cluster partition of \bar{T} . Node w , on the other hand, may be of any type. We check whether v is an ancestor of w in \bar{T} according to Lemma 3.2. Let $dfs(v)$ and $dfs(w)$ be the DFS labels of $a(v)$ and $a(w)$, respectively, in the DFS labeling of \bar{T}^m . We extract $dfs(v)$ and $dfs(w)$ from the blocks a_j and b_j of $L(v)$ and $L(w)$, respectively. Then we can determine whether the representative $a(v)$ of v in \bar{T}^m and the representative $a(w)$ of w in \bar{T}^m are equal, and whether $a(v)$ is an ancestor of $a(w)$, by comparing $dfs(v)$ and $dfs(w)$.

If $dfs(v)$ is equal to $dfs(w)$, then $a(v)$ is the same node as $a(w)$. We know that

$a(v)$ is either a spine node or a boundary node. Since a boundary node v is not $a(w)$ for any $w \neq v$, and since $a(v) = a(w)$, it follows that v and w are spine nodes. Thus, by Lemma 3.2, $v \prec_T w$ if and only if w is a descendant of v on their spine path, i.e., if and only if $v <_\alpha w$. Let $c(v)$ and $c(w)$ be the alphabetic codes in blocks a_j and b_j of v and w , respectively. By the definition of an alphabetic code and the definition of the labels, we have that $v <_\alpha w$ if and only if $c(v) <_{\text{lex}} c(w)$. Thus in this case we answer that $v \prec_T w$ if and only if $c(v) <_{\text{lex}} c(w)$.

If $dfs(v)$ is not equal to $dfs(w)$, then $a(v)$ is not equal to $a(w)$. By Lemma 3.2, a necessary condition for v to be an ancestor of w is that $a(v)$ be an ancestor of $a(w)$ in \overline{T}^m , so we check that using $dfs(v)$ and $dfs(w)$. Furthermore, by Lemma 3.2, if $a(v)$ is indeed an ancestor of $a(w)$ in \overline{T}^m , then we can conclude that $v \prec_T w$ except when v and w are in the same cluster, v is a spine node, and w is an internal clustered node. In this case $a(v) = s(u_1, u_2)$ is the parent of $a(w) = \ell(u_1, u_2)$, where u_1 and u_2 are the boundary nodes of $C(v) = C(w)$. Fortunately, the DFS labeling scheme allows us to determine whether $a(w)$ is indeed a child of $a(v)$ and thereby identify this case where v is a spine node and w is an internal clustered node of the same cluster. First we check whether $a(w)$ is indeed a leaf, by verifying that $dfs(w)$ consists only of a single number. Assume this is indeed the case, let $k = dfs(w)$, and let $(first, last) = dfs(v)$. Then $a(v)$ is a leaf child of $a(w)$ if and only if $k = first + 1$ or $last = k + 1$.

If indeed v is a spine node and w is an internal clustered node of some cluster, then, by Lemma 3.2, $v \prec_T w$ if and only if $sub(w)$ is a descendant of v on v 's spine path. Let $c(v)$ and $c(w)$ be the alphabetic codes in blocks a_j and b_j of $L(v)$ and $L(w)$, respectively. By the definition of the labels and the definition of an alphabetic code, $sub(w)$ is a descendant of v on v 's spine path if and only if $c(v) = c(w)$ or $c(v) <_{\text{lex}} c(w)$.

Since the decoder does not know the length of $c(w)$ it cannot easily extract it. However, since $c(w)$ is not a proper prefix of $c(v)$ (as $c(w)$ and $c(v)$ are strings in an alphabetic code) we have that $c(v) \leq_{\text{lex}} c(w)$ if and only if $c(v)$ is lexicographically smaller than the suffix of the label of w starting from and including $c(w)$. Thus in case v is a spine node and w is an internal clustered node of some cluster of \overline{T} , the second phase concludes that $v \prec_T w$ if and only if $c(v)$ is lexicographically smaller than the suffix of the label of w starting from and including $c(w)$.

Note that the above test for the ancestor relation using the DFS labeling and alphabetic codes uses a constant number of basic and fast RAM operations, i.e., bit wise AND/OR, left/right shifts, and less-than comparisons. Our labeling scheme avoids the sometimes more costly operations such as multiplication, retrieval of the most significant bit, or nonstandard operations precomputed and stored in a pre-computed table.

5. The length of the labels. In this section we give an upper bound on the length of $L(v)$ for any $v \in T$. We first bound the length of any block which is not the last block.

LEMMA 5.1. *Let $L(v) = b_1 b_2, \dots, b_k$ for some $k \geq 2$. Then,*

$$b_1 \leq \log \left(\frac{n}{|T(b_1)|} \right) + O(1),$$

$$\text{and for } 1 < i < k, \quad |b_i| \leq \log \left(\frac{|T(b_1, \dots, b_{i-1})|}{|T(b_1, \dots, b_{i-1} b_i)|} \right) + O(1).$$

Proof. Since b_i is not the last block in $L(v)$, v is either an internal clustered node or a leaf clustered node in $T(b_1, \dots, b_{i-1})$ or in T in case $i = 1$. In case v is an

internal clustered node, the claim follows from the definition of the label of an internal clustered node and the definition of $|T(b_1, \dots, b_{i-1}b_i)|$ (see section 3.2). In case v is a leaf clustered node, then by the definition of the label we have that $|b_i| \leq \log x + O(1)$. But in this case we also have that $|T(b_1, \dots, b_{i-1}b_i)| \leq \lceil |T(b_1, \dots, b_{i-1})|/x \rceil$ by the definition of a cluster partition. \square

The following lemma gives a bound on the length of the last block.

LEMMA 5.2. *Let $L(v) = b_1b_2, \dots, b_k$. Then*

$$|b_k| \leq \log(|T(b_1, \dots, b_{k-1})|) + O(\log x) + O(1).$$

Proof. Since b_k is the last block, v is either a boundary node or a spine node in $T(b_1, \dots, b_{k-1})$. If v is a boundary node, then $|b_k| \leq 2 \log x + O(1)$. If v is a spine node, then $|b_k| \leq \log(|T(b_1, \dots, b_{k-1})|) + \log x + O(1)$. \square

Last we bound the number of blocks.

LEMMA 5.3. *Let $L(v) = b_1b_2, \dots, b_k$. Then $k \leq \frac{\log n}{\log x}$.*

Proof. By the definition of a cluster partition we have for $i < k$ that $|T(b_1, \dots, b_{i-1})|/|T(b_1, \dots, b_i)| \geq x$. Since $|T(\epsilon)| = |T| = n$ the lemma follows. \square

Now from these three lemma we immediately obtain the following result.

LEMMA 5.4. *For every $v \in T$, $|L(v)| \leq \log n + O(\log x) + O(\frac{\log n}{\log x})$.*

Hence, choosing $x = 2^{\sqrt{\log n}}$, we obtain the claimed bound of $\log n + O(\sqrt{\log n})$ for the worst-case length of a label in T .

6. Preprocessing time. It is rather straightforward to compute the labels in time $O(n\sqrt{\log n})$, spending linear time calculating the part of the labels in each of the $\sqrt{\log n}$ recursive levels. In each recursive level we calculate the cluster partition in each of the trees that get labeled in that level, and we also calculate alphabetic code for the nodes on every spine path.

We can improve this preprocessing time to linear by slightly changing the labeling scheme, while keeping the labels of length $\log n + O(\sqrt{\log n})$. We do that as follows. We obtain a cluster partition of T with $x = n/\sqrt{\log n}$, as described in section 2. Hence the clusters are of size at most $\sqrt{\log n}$. For this cluster partition we construct a tree T' that is similar to T^m defined in section 3.1. We define T' as follows. For each cluster with two boundary nodes v and w , we have in T' the nodes $v, w, s(v, w)$ and two edges $(v, s(v, w))$ and $(w, s(v, w))$. For each cluster C with one boundary node v , we have in T' the nodes v and $\ell(v, C)$ and the edge $(v, \ell(v, C))$. (In contrast with T^m , in T' we do not have a leaf $\ell(v, w)$ for a cluster with boundary nodes v and w .) By Lemma 3.1, the tree T' consists of $O(n/\sqrt{\log n})$ nodes. We label T' by the labeling scheme of section 3, using the naive labeling algorithm mentioned above. Since the size of T' is $O(n/\sqrt{\log n})$, computing this labeling of T' takes linear time. We denote the label of a node v in T' by $L'(v)$.

We also label the nodes of each cluster separately for ancestor queries using the *DFS* labeling scheme. This takes linear time for all clusters. We denote the label of v in its cluster by $CL(v)$.

For each node v in T we associate a node in T' , denoted by $b(v)$. (This mapping is similar to the mapping $a()$ from section 3.1.) If v is a boundary node, then $b(v) = v$. If v is a spine node or an internal clustered node in a cluster with two boundary nodes u, w , then $b(v) = s(u, w)$. If v is a leaf clustered node in a cluster C with boundary node w , then $b(v) = \ell(w, C)$.

We define the label of a node $v \in T$ to be $L(v) = Q(v) \cdot CL(v) \cdot L'(b(v))$, where $Q(v)$ are two bits that determine the type of the node with respect to the cluster partition

(see section 3.2). Since $|L'(b(v))| \leq \log n + O(\sqrt{\log n})$ and $|CL(v)| \leq O(\log \log n)$ we obtain that $|L(v)| \leq \log n + O(\sqrt{\log n})$.

To test whether v is an ancestor of w in T we proceed as follows. If v is a boundary node, we check whether $b(v)$ is an ancestor of $b(w)$. If v is a spine node and $b(v) \neq b(w)$, we test whether $b(v)$ is ancestor of $b(w)$. If v is a spine node and $b(v) = b(w)$, we use $CL(v)$ and $CL(w)$ to test whether v is ancestor of w . In the remaining cases, for v to be an ancestor of w , $b(v)$ must be equal to $b(w)$, and if that is indeed the case, then we check that v is an ancestor of w in their cluster using $CL(v)$ and $CL(w)$.

We conclude this section with the following theorem, which is the main result of this paper.

THEOREM 6.1. *We can label the nodes of a tree of size n in $O(n)$ time by labels of length at most $\log n + O(\sqrt{\log n})$ such that, given the labels of two nodes v and w , we can determine whether v is an ancestor of w in T in $O(1)$ time.*

7. Conclusions. We have described a labeling scheme for ancestor queries, where the length of each label is bounded by $\log n + O(\sqrt{\log n})$ bits. The labeling is computed in linear time, and an ancestor query takes $O(1)$ time. As we mentioned, Alstrup, Bille, and Rauhe [3] proved that for every labeling scheme there is a tree that requires a label of length $\log n + \Omega(\log \log n)$ bits. Closing this gap between the upper and the lower bounds is an intriguing open question.

REFERENCES

- [1] S. ABITEBOUL, P. BUNEMAN, AND D. SUCIU, *Data on the Web: From Relations to Semistructured Data and XML*, Morgan-Kaufmann, San Francisco, 1999.
- [2] S. ABITEBOUL, H. KAPLAN, AND T. MILO, *Compact labeling schemes for ancestor queries*, in Proceedings of the Twelfth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA), Washington, DC, 2001, SIAM, Philadelphia, 2001, pp. 547–556.
- [3] S. ALSTRUP, P. BILLE, AND T. RAUHE, *Labeling schemes for small distances in trees*, in Proceedings of the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA), Baltimore, MD, 2003, SIAM, Philadelphia, 2003, pp. 689–698.
- [4] S. ALSTRUP, C. GAVOILLE, H. KAPLAN, AND T. RAUHE, *Nearest common ancestors: A survey and a new algorithm for a distributed environment*, Theory Comput. Syst., 37 (2004), pp. 441–456.
- [5] S. ALSTRUP, J. HOLM, K. DE LICHTENBERG, AND M. THORUP, *Minimizing diameters of dynamic trees*, in Automata, Languages and Programming, 24th International Colloquium (ICALP), Lecture Notes in Comput. Sci. 1256, Springer-Verlag, New York, 1997, pp. 270–280.
- [6] S. ALSTRUP, J. HOLM, AND M. THORUP, *Maintaining median and center in dynamic trees*, in Proceedings of the 7th Scandinavian Workshop on Algorithm Theory (SWAT), Lecture Notes Comput. Sci. 1851, Springer-Verlag, New York, 2000, pp. 46–56.
- [7] S. ALSTRUP AND T. RAUHE, *Improved labeling scheme for ancestor queries*, in Proceedings of the Thirteen Annual ACM-SIAM Symposium on Discrete Algorithms (SODA), San Francisco, CA, 2002, SIAM, Philadelphia, 2002, pp. 947–953.
- [8] *Altavista*, online search engine at <http://www.altavista.com>.
- [9] M. A. BREUER, *Coding vertexes of a graph*, IEEE Trans. Inform. Theory, IT-12 (1966), pp. 148–153.
- [10] M. A. BREUER AND J. FOLKMAN, *An unexpected result on coding vertices of a graph*, J. Math. Anal. Appl., 20 (1967), pp. 583–600.
- [11] T. H. CORMEN, C. E. LEISERSON, AND R. L. RIVEST, *Introduction to Algorithms*, MIT Press/McGraw-Hill, New York, 1991.
- [12] A. DEUTSCH, M. F. FERNANDEZ, D. FLORESCU, A. Y. LEVY, AND D. SUCIU, *A query language for XML*, Computer Networks, 31 (1999), pp. 1155–1169.
- [13] G. N. FREDERICKSON, *Ambivalent data structures for dynamic 2-edge-connectivity and k smallest spanning trees*, SIAM J. Comput., 26 (1997), pp. 484–538.
- [14] C. GAVOILLE AND D. PELEG, *Compact and localized distributed data structures*, J. Distributed Comput., 16 (2003), pp. 111–120.

- [15] C. GAVOILLE, D. PELEG, S. PÉRENNÈS, AND R. RAZ, *Distance labeling in graphs*, J. Algorithms, 53 (2004), pp. 85–112.
- [16] E. N. GILBERT AND E. F. MOORE, *Variable-length binary encodings*, Bell System Technical Journal, 38 (1959), pp. 933–967.
- [17] *Google*, online search engine at <http://www.google.com/>.
- [18] S. KANNAN, M. NAOR, AND S. RUDICH, *Implicit representation of graphs*, SIAM J. Discrete Math., 5 (1992), pp. 596–603.
- [19] H. KAPLAN AND T. MILO, *Short and simple labels for small distances and other functions*, in Proceedings of the 7th Workshop on Algorithms and Data Structures, Providence, RI, 2001, Lecture Notes in Comput. Sci. 2125, Springer, New York, 2001, pp. 246–257.
- [20] H. KAPLAN, T. MILO, AND R. SHABO, *A comparison of labeling schemes for ancestor queries*, in Proceedings of the Thirteenth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA), San Francisco, CA, 2002, SIAM, Philadelphia, 2002, pp. 954–963.
- [21] M. KATZ, N. A. KATZ, A. KORMAN, AND D. PELEG, *Labeling schemes for flow and connectivity*, SIAM J. Comput., 34 (2004), pp. 23–40.
- [22] M. KATZ, N. A. KATZ, AND D. PELEG, *Distance labeling schemes for well-separated graph classes*, Discrete Appl. Math., 145 (2005), pp. 384–402.
- [23] Q. LI AND B. MOON, *Indexing and querying xml data for regular path expression*, in Proceedings of the 27th International Conference on Very Large Data Bases (VLDB), Rome, 2001, Morgan Kaufmann, San Francisco, 2001, pp. 361–370.
- [24] K. MEHLHORN, *A best possible bound for the weighted path length of binary search trees*, SIAM J. Comput., 6 (1997), pp. 235–239.
- [25] OASIS, *Organization for the Advancement of Structures Information Standards*, available online at <http://www.oasis-open.org/>.
- [26] D. PELEG, *Proximity-preserving labeling schemes*, J. Graph Theory, 33 (2000), pp. 167–176.
- [27] N. SANTORO AND R. KHATIB, *Labeling and implicit routing in networks*, Comput. J., 28 (1985), pp. 5–8.
- [28] R. TARJAN, *Depth-first search and linear graph algorithms*, SIAM J. Comput., 1 (1972), pp. 146–160.
- [29] M. THORUP AND U. ZWICK, *Compact routing schemes*, in Proceedings of the 13th Annual ACM Symposium on Parallel Algorithms and Architectures, Crete, Greece, 2001, ACM, New York, 2001, pp. 1–10.
- [30] A. K. TSAKALIDIS, *Maintaining order in a generalized linked list*, Acta Inform., 21 (2004), pp. 101–112.
- [31] W3. *Extensible Markup Language (XML) 1.0*, documentation online at <http://www.w3.org/XML>.
- [32] W3C. *Extensible Stylesheet Language (XSL)*, documentation online at <http://www.w3.org/Style/XSL/>.
- [33] *HyperText Markup Language (HTML)*, documentation online at <http://www.w3.org/MarkUp>.
- [34] *Xyleme. A dynamic data warehouse for the XML data of the web*, documentation online at <http://www.xyleme.com>.

QUANTUM QUERY COMPLEXITY OF SOME GRAPH PROBLEMS*

CHRISTOPH DÜRR[†], MARK HEILIGMAN[‡], PETER HØYER[§], AND MEHDI MHALLA[¶]

Abstract. Quantum algorithms for graph problems are considered, both in the adjacency matrix model and in an adjacency list-like array model. We give almost tight lower and upper bounds for the bounded error quantum query complexity of CONNECTIVITY, STRONG CONNECTIVITY, MINIMUM SPANNING TREE, and SINGLE SOURCE SHORTEST PATHS. For example, we show that the query complexity of MINIMUM SPANNING TREE is in $\Theta(n^{3/2})$ in the matrix model and in $\Theta(\sqrt{nm})$ in the array model, while the complexity of CONNECTIVITY is also in $\Theta(n^{3/2})$ in the matrix model but in $\Theta(n)$ in the array model. The upper bounds utilize search procedures for finding minima of functions under various conditions.

Key words. graph theory, quantum algorithm, lower bound, connectivity, minimum spanning tree, single source shortest paths

AMS subject classifications. 81-04, 68W20, 68R10, 05C85

DOI. 10.1137/050644719

1. Introduction. A primary goal of the theory of quantum complexity is to determine when quantum computers may offer a computational speed-up over classical computers. Today there are only a few results which give a polynomial time quantum algorithm for some problem for which no classical polynomial time solution is known. We are interested in studying the potentials for speed-up for problems for which there already are efficient classical algorithms. Basic graph problems are interesting candidates.

We study the query complexity of these problems, meaning the minimal number of queries to the graph required for solving the problem. Throughout this paper, the symbol $[n]$ denotes the set $\{0, 1, \dots, n-1\}$. We consider two query models for directed graphs:

- *the adjacency matrix model*, where the graph is given as the adjacency matrix $M \in \{0, 1\}^{n \times n}$, with $M_{ij} = 1$ if and only if $(v_i, v_j) \in E$, and
- *the adjacency array model*, where we are given the out-degrees of the vertices d_1^+, \dots, d_n^+ and for every vertex u an array with its neighbors $f_i : [d_i^+] \rightarrow [n]$.

*Received by the editors September 1, 2004; accepted for publication (in revised form) October 7, 2005; published electronically March 24, 2006. This paper subsumes the three manuscripts “A quantum algorithm for finding the minimum” (<http://arxiv.org/abs/quant-ph/9607014>), “Quantum algorithms for lowest weight paths and spanning trees in complete graphs” ([quant-ph/0303131](http://arxiv.org/abs/quant-ph/0303131)), and “Quantum query complexity of graph connectivity” ([quant-ph/0303169](http://arxiv.org/abs/quant-ph/0303169)), and includes several other previously unpublished results. We are grateful to Yaohui Lei for his permission to include results presented in [quant-ph/0303169](http://arxiv.org/abs/quant-ph/0303169) in this paper.

<http://www.siam.org/journals/sicomp/35-6/64471.html>

[†]CNRS, Laboratoire d’Informatique de l’Ecole Polytechnique, Palaiseau, France (durr@lix.polytechnique.fr). This author’s work conducted while he was affiliated with LRI, Université Paris-Sud, Orsay, France. This author’s research was partially supported by the EU fifth framework program RESQ IST-2001-37559 and RAND-APX IST-1999-14036, by grant CNRS/STIC 01N80/0502, and by grant ACI Cryptologie CR/02 02 0040 of the French Research Ministry.

[‡]Advanced Research and Development Activity, Suite 6644, National Security Agency, 9800 Savage Road, Fort Meade, MD 20755 (miheili@nsa.gov).

[§]Department of Computer Science, University of Calgary, Alberta, Canada (hoyer@cpsc.ucalgary.ca). This author’s work was supported in part by the Alberta Ingenuity Fund, the Pacific Institute for the Mathematical Sciences, the Canadian Institute for Advanced Research, and the Natural Sciences and Engineering Research Council of Canada.

[¶]CNRS, Laboratoire Leibniz, Institut IMAG, Grenoble, France (Mehdi.Mhalla@imag.fr).

So $f_i(j)$ returns the j th neighbor of vertex i , according to some arbitrary but fixed numbering of the outgoing edges of i . In this paper the upper bounds for this model are all at least n , so we assume henceforth that the degrees are given as part of the input and we count only queries to the arrays f_i . In addition the arrays satisfy the *simple graph* promise

$$(1.1) \quad \forall i \in [n], j, j' \in [k], j \neq j' : f_i(j) \neq f_i(j')$$

ensuring the graph is not a multigraph, i.e., does not have multiple edges between any two vertices.

For undirected graphs we require an additional promise on the input, namely, that M is symmetric in the matrix model, and for the array model that for all $i, i' \in [n]$, if $\exists j \in [k] : f_i(j) = i'$, then $\exists j' \in [k] : f_{i'}(j') = i$. Note that in the matrix model this symmetry assumption does not create a promise problem on undirected graphs since we may assume that the input is upper triangular.

Weighted graphs are encoded by a weight matrix, where for convenience we set $M_{ij} = \infty$ if $(v_i, v_j) \notin E$. In the adjacency array model, the graph is encoded by a sequence of functions $f_i : [d_i^+] \rightarrow [n] \times \mathbb{N}$, such that if $f_i(j) = (i', w)$, then there is an edge $(v_i, v_{i'})$ and it has weight w .

We emphasize that the array model is different from the standard list model. In the latter, we have access to the neighbors of a given vertex only as a list, and thus querying the i th neighbor requires i accesses to the list. This is also true on a quantum computer, so its speed-up is quite restricted.

Many other query models are of course possible; for example, we could be given an array of edges $f : [m] \rightarrow [n] \times [n]$ or an ordered array (which is up to $O(n)$ preprocessing the same as the adjacency array model). For simplicity, we use the array model as presented above.

For the quantum query complexity of general monotone graph properties, a lower bound of $\Omega(\sqrt{n})$ is known in the matrix model, as shown by Buhrman et al. [9]. We are not aware of any quantum or classical lower bounds in the array model.

In this paper we show that the quantum query complexity of CONNECTIVITY is $\Theta(n^{3/2})$ in the matrix model and $\Theta(n)$ in the array model. The classical randomized query complexity of CONNECTIVITY in the matrix model is $\Omega(n^2)$ by a sensitivity argument: Distinguishing the graph consisting of two length $n/2$ paths from the graph consisting of those two paths, plus an additional edge connecting them, requires $\Omega(n^2)$ queries.

We study the complexity of three other problems; see Table 1.1. In STRONG CONNECTIVITY we are given a directed graph and have to decide if there is a directed path between every pair of vertices. In MINIMUM SPANNING TREE we are given a weighted graph and have to compute a spanning tree with minimal total edge weight. In SINGLE SOURCE SHORTEST PATHS we have to compute the shortest paths according to the total edge weight from a given source vertex to every other vertex. The quantum query complexity of these three problems is $\Omega(n^{3/2})$ in the matrix model and $\Omega(\sqrt{nm})$ in the array model. We give almost tight upper bounds.

We note that for graphs with a large number of edges ($m = \Theta(n^2)$), the complexities are (almost) the same in the matrix and array models for all problems but CONNECTIVITY. However, the models still differ in that case. For example, the test $(u, v) \in E$ costs a single query in the matrix model and $\Theta(\sqrt{\min\{d_u^+, d_v^+\}})$ queries in the array model since we do not assume any order on the arrays f_u and f_v .

TABLE 1.1
Quantum query complexity of some graph problems.

Problem	Matrix model	Array model
MINIMUM SPANNING TREE	$\Theta(n^{3/2})$	$\Theta(\sqrt{nm})$
CONNECTIVITY	$\Theta(n^{3/2})$	$\Theta(n)$
STRONG CONNECTIVITY	$\Theta(n^{3/2})$	$\Omega(\sqrt{nm}), O(\sqrt{nm \log n})$
SINGLE SRC. SHORT. PATHS	$\Omega(n^{3/2}), O(n^{3/2} \log^2 n)$	$\Omega(\sqrt{nm}), O(\sqrt{nm \log^2 n})$

The time complexities of the algorithms are the same as the query complexities up to log-factors. The algorithms given for CONNECTIVITY and STRONG CONNECTIVITY can be altered to also output the (strongly) connected components without increasing the asymptotic complexity. The space requirement is $O(\log n)$ qubits and $O(n \log n)$ classical bits. If we constrain the space (both classical and quantum) to $O(\log n)$ qubits, the problems may be solved by random walks. Quantum random walks have been the subject of several papers [1, 10, 15], in particular for the *st*-CONNECTIVITY problem [22].

Other work on the query complexity of graph problems has been done independently of us. For example, [4] shows that testing whether a given graph is bipartite needs $\Omega(n^{3/2})$ queries in the matrix model. Note that a graph is bipartite if and only if it does not contain an odd cycle. For the array model a lower bound can be constructed from our lower bound for connectivity, showing that $\Omega(\sqrt{nm})$ queries are necessary for any bounded error quantum algorithm which distinguishes a single even cycle from two disjoint odd cycles. Matching upper bounds follow from our connectivity algorithms: simply construct a spanning forest of the graph, color the nodes of each tree alternately black and white, and finally use the quantum search procedure to find an edge with endpoints of the same color. Such an edge creates an odd cycle and exists if and only if the graph contains an odd cycle.

2. Tools used in this paper. We use two fundamental tools. The first is amplitude amplification [7, 8], which we use when proving the upper bounds; the second is Ambainis’s lower bound technique [2].

Amplitude amplification is a generalization of Grover’s search algorithm [13]. Since it is the most important tool used in our algorithms, we restate the exact results we require. We are given a boolean function F defined on a domain of size n . The function is given as a black box so that the only way we can obtain information about F is via evaluating F on elements in the domain. The search problem considered by Grover is to find an element x for which $F(x) = 1$, provided one exists. We say that x is a *solution* to the search problem and that x is *good*. We use three generalizations of the search algorithm—all of which we refer to as “the search algorithm.”

- If there are t elements mapped to 1 under F , with $t > 0$, the search algorithm returns a solution after an expected number of at most $\frac{9}{10} \sqrt{n/t}$ queries to F . The output of the algorithm is chosen uniformly at random among the t solutions. The algorithm does not require prior knowledge of t [6].
- A second version uses $O(\sqrt{n})$ queries to F in the worst case and outputs a solution with probability at least a constant, provided there is one [6].
- A third version uses $O(\sqrt{n \log 1/\epsilon})$ queries to F and finds a solution with probability at least $1 - \epsilon$, provided there is one [9].

We note that for very sparse graphs given in the adjacency matrix model, it may for some applications be efficient to initially learn all entries of the matrix by reiterating the first version of the search algorithm, for instance, as formalized in

Fact 2.1.

FACT 2.1. *Let k be given. There is a quantum algorithm that takes as input an $n \times n$ boolean matrix M , uses $O(n\sqrt{k})$ queries to M , and outputs a set S of 1-entries of M so that with probability at least $\frac{1}{2}$, S is of cardinality at least k or contains all 1-entries of M in case M has less than k 1-entries.*

Our lower bounds use a technique introduced by Ambainis [2].

THEOREM 2.2 (see [2, Theorem 6]). *Let $L \subseteq \{0, 1\}^*$ be a decision problem. Let $X \subseteq L$ be a set of positive instances and $Y \subseteq \bar{L}$ a set of negative instances. Let $R \subseteq X \times Y$ be a relation between instances of the same size. Let values $m, m', \ell_{x,i}$, and $\ell'_{y,i}$ for $x, y \in \{0, 1\}^n$ and $x \in X, y \in Y, i \in [n]$ be so that*

- *for every $x \in X$ there are at least m different $y \in Y$ in relation with x ,*
- *for every $y \in Y$ there are at least m' different $x \in X$ in relation with y ,*
- *for every $x \in X$ and $i \in [n]$ there are at most $\ell_{x,i}$ different $y \in Y$ in relation with x which differ from x at entry i ,*
- *for every $y \in Y$ and $i \in [n]$ there are at most $\ell'_{y,i}$ different $x \in X$ in relation with y which differ from y at entry i .*

Then the quantum query complexity of L is $\Omega(\sqrt{mm'/\ell_{\max}})$, where ℓ_{\max} is the maximum of $\ell_{x,i}\ell'_{y,i}$ subject to xRy and $x_i \neq y_i$.

3. Minima finding. Many graph problems are optimization problems, as are finding a minimum spanning tree, single source shortest paths, and largest connected components. Most quantum algorithms for such optimization problems utilize the search algorithm discussed above. A very basic and abstract optimization problem is as follows. Suppose we are given a function f defined on a domain of size n , and we want to find an index i so that $f(i)$ is a minimum in the image of f . This minimization problem was considered in [11], which gives an optimal quantum algorithm that uses $O(\sqrt{n})$ queries to f and finds such an i with constant probability. Since it will make the rest of the section easier to understand, we include it here.

1. Initially let $j \in [N]$ be an index chosen uniformly at random.
2. Repeat forever
 - (a) Find an index $i \in [N]$ such that $f(i) < f(j)$.
 - (b) Set $j := i$.

The analysis of this minimization algorithm is straightforward.

THEOREM 3.1 (see [11]). *The expected number of queries to f , until j contains the index of a minimum in the image of f , is $O(\sqrt{N})$.*

Proof. Without loss of generality assume that f is injective. Every index $j \in [N]$ has a *rank*, which we define as the number of indices i such that $f(i) \leq f(j)$. Consider an iteration of the main loop, and suppose the rank of j is r in Step 2(a). Let t be a power of two such that $t \leq r < 2t$. After an expected number of at most $\frac{9}{10}\sqrt{N}/t$ queries, the algorithm finds an index i of rank $r_i < r$. Since the algorithm in Step 2(b) picks an index of rank smaller than j uniformly at random, $r_i < t$ with probability at least $\frac{1}{2}$. Thus after an expected number of at most $\frac{9}{5}\sqrt{N}/t$ queries, we have found an index of rank less than t .¹ Applying this argument repeatedly, the expected total number of queries is at most $\frac{9}{5} \sum_{k \geq 0} \sqrt{N}/2^k$. Since the geometric summation $\sum_{k \geq 0} \frac{1}{\sqrt{2^k}}$ is upper bounded by a constant, the expected total number of queries is $O(\sqrt{N})$. \square

¹In fact, the probability that the index of rank r is picked at some point during the run of the algorithm is exactly $\frac{1}{r}$.

Let $c'\sqrt{N}$ be the expected number of queries to f until j contains the index of a minimum. Stopping the algorithm after $2c'$ queries gives a quantum algorithm with error probability upper bounded by $\frac{1}{2}$.

For the purposes of this paper, we require the following generalizations of the minimum finding problem, illustrated in Figure 3.1.

PROBLEM 1 (find d smallest values of a function). *Let \mathbb{N}^* denote $\mathbb{N} \cup \{\infty\}$. Given a function $f : [N] \rightarrow \mathbb{N}^*$ and an integer $d \in [N]$, we wish to find d distinct indices mapping to smallest values, i.e., a subset $I \subseteq [N]$ of cardinality d such that for any $j \in [N] \setminus I$ we have that $f(i) \leq f(j)$ for all $i \in I$.*

In the rest of this section, we assume $d \leq N/2$. In the following problem we are given a different function $g : [N] \rightarrow \mathbb{N}$, such that $g(j)$ defines the *type* of j . Let $e = |\{g(j) : j \in [N]\}|$ be the number of different types.

PROBLEM 2 (find d elements of different type). *Given a function g and an integer d' we wish to find integer $d = \min\{d', e\}$ and a subset $I \subseteq [N]$ of cardinality d such that $g(i) \neq g(i')$ for all distinct $i, i' \in I$.*

Now we present a generalization of both problems.

PROBLEM 3 (find d smallest values of different type). *Given two functions f, g and an integer d' we wish to find integer $d = \min\{d', e\}$ and a subset $I \subseteq [N]$ of cardinality d such that $g(i) \neq g(i')$ for all distinct $i, i' \in I$ and such that for all $j \in [N] \setminus I$ and $i \in I$, if $f(j) < f(i)$, then $f(i') \leq f(j)$ for some $i' \in I$ with $g(i') = g(j)$.*

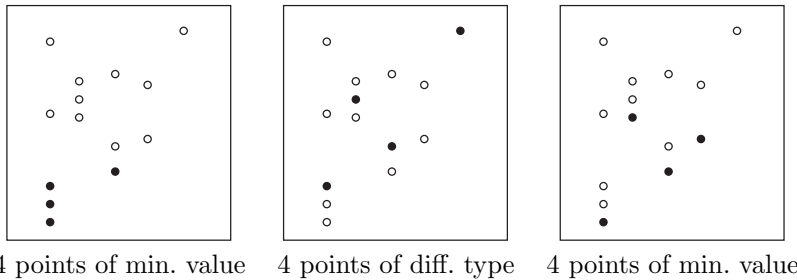


FIG. 3.1. Illustration of the three problems. Each index i is illustrated by a point with horizontal coordinate $g(i)$ and vertical coordinate $f(i)$.

It is clear that Problems 1 and 2 are special cases of Problem 3. In this section, we give an upper bound of $O(\sqrt{dN})$ for Problem 3. In section 8, we then show a lower bound of $\Omega(\sqrt{dN})$ for Problems 1 and 2, implying that all three problems are of complexity $\Theta(\sqrt{dN})$. We prove the upper bound by a simple greedy algorithm. Consider a subset $I \subseteq [N]$ of d indices of different types. We say an index $j \in [N]$ is good for I if

1. either $g(j) = g(i)$ and $f(j) < f(i)$ for some $i \in I$,
2. or $g(j) \notin g(I)$ and $f(j) < f(i)$ for some $i \in I$.

In the former case we say j is a good index of *known type* and in the latter that j is a good index of *unknown type*. In each iteration of the greedy algorithm, we find a good index j by the search algorithm and then improve I by replacing some index in I by j .

1. Initially, let $I = \{N + 1, \dots, N + d'\}$ be a set of artificial indices of unique different types and unique maximal value.
2. Repeat forever

- (a) Let t denote the number of good elements for I . (Note: This step is not required but is only included for the purpose of simplifying the analysis of the algorithm.)
- (b) Use the first version of the search algorithm to find a good element $j \in [N]$ for I .
- (c) Set $I = \text{improve}(I, j)$ where we improve I by replacing with j the element in I that has the same type as j if j is of known type, and by replacing with j some element in I with largest f -value if j is of unknown type.

The next lemma shows that we need only an expected number of $O(d)$ iterations of the main loop to eliminate a constant fraction of the remaining good elements.

LEMMA 3.2. *Let $I \subseteq [N]$ be any subset of d' indices of different types with $t > 0$ good elements of e types. After an expected number of $O(d)$ iterations of the main loop there are at most $\frac{3}{4}t$ good elements for I . Here $d = \min\{d', e\}$.*

Proof. For notational simplicity assume f is injective. Set $I_0 = I$ and let $T_0 = T$ be the set of good elements for I . Let T_j denote the set of good elements after j iterations of the main loop for $j > 0$. Similarly, let I_j denote the selected index set after j iterations for $j > 0$. Set $t_k = |T_k|$. In particular, $I_0 = I$ and $t_0 = t$. Let y_{mid} denote the $\lfloor t/2 \rfloor$ th smallest of the t elements according to f . For any subset $S \subseteq [N + d']$, let $\text{low}(S)$ denote the number of elements in S that are no bigger than y_{mid} according to f .

Note that initially

- $\text{low}(T_0) = \lfloor t/2 \rfloor$ and
- $\text{low}(I_0) < d$.

By the nature of the greedy algorithm, $\text{low}(T_{k+1}) \leq \text{low}(T_k)$ and $\text{low}(I_{k+1}) \geq \text{low}(I_k)$ for any $k \geq 0$. Note that

- if $\text{low}(T_k) < \frac{t}{4}$, then we have eliminated at least a fraction of $\frac{1}{4}$ of the initially t good elements for I , and similarly,
- if $\text{low}(I_k) = d$, then we have eliminated at least a fraction of $\frac{1}{2}$ of the initially t good elements for I .

We claim that in each iteration of the main loop, as long as $\text{low}(T_k) \geq \frac{t}{4}$, with probability at least $\frac{1}{32}$, at least one of the following two events happens:

- $\text{low}(T_{k+1}) \leq \text{low}(T_k) \left(1 - \frac{1}{32d}\right)$,
- $\text{low}(I_{k+1}) = \text{low}(I_k) + 1$.

Assume $\text{low}(T_k) \geq \frac{t}{4}$, since otherwise we are done. Consider the element j picked in Step 2(b). First suppose the majority of the $\text{low}(T_k)$ indices are of unknown type with respect to I_k . Then, with probability at least $\frac{1}{8}$, index j is among these, in which case $\text{low}(I_{k+1}) = \text{low}(I_k) + 1$.

Now suppose the majority of the $\text{low}(T_k)$ indices are of known type with respect to I_k . Then, with probability at least $\frac{1}{8}$, index j is among these. Conditioned on this happening, with probability at least $\frac{1}{2}$, there are at least $\frac{\text{low}(T_k)}{4d}$ good elements for I_k of the same type as j . With probability at least $\frac{1}{2}$, at least half of these are not good for I_{k+1} . Thus, with probability at least $\frac{1}{32}$, we have eliminated at least $\frac{t}{32d}$ of the remaining elements in T_j .

This proves the claim. It follows that after an expected number of $O(d)$ iterations of the main loop, we have eliminated at least a fraction of $\frac{1}{4}$ of the initially t good elements. \square

The above lemma implies that, for $t > 2d$, after an expected number of $O(d\sqrt{N/t})$ applications of function f , the number of good elements is at most $\frac{t}{2}$. Hence, for any

$t > 2d$, the expected number applications of function f required until we have that $t \leq 2d$ for the first time is on the order of

$$d \left(\sqrt{\frac{N}{d}} + \sqrt{\frac{N}{2d}} + \sqrt{\frac{N}{4d}} + \sqrt{\frac{N}{8d}} + \dots \right) \in O(\sqrt{dN}).$$

Once $t \leq 2d$ for the first time, the expected number of applications of f required before $t = 0$ for the first time is on the order of $\sum_{j=1}^{2d} \sqrt{N/j}$, which is $O(\sqrt{dN})$.

COROLLARY 3.3. *In the greedy algorithm given above, after an expected number of $O(\sqrt{dN})$ applications of function f , there are no good elements for I ; that is, $t = 0$.*

The next theorem follows immediately.

THEOREM 3.4. *The problem FIND d SMALLEST VALUES OF DIFFERENT TYPE has bounded error quantum query complexity $O(\sqrt{dN})$.*

Nayak and Wu give in [19] a bounded error quantum algorithm that, given a function $f : [N] \rightarrow \mathbb{N}$ and two integers d and Δ , outputs an index i such that the rank of $f(i)$ is between $d - \Delta$ and $d + \Delta$. The query complexity of their algorithm is $O(M \log M \log \log M)$, where $M = \sqrt{N/\Delta} + \sqrt{d(N-d)}/\Delta$. Setting $\Delta = \frac{1}{2}$, it would find the d th smallest element with $O(\sqrt{dN} \log N \log \log N)$ queries. Nayak [18] later improved this algorithm to $O(\sqrt{dN})$, matching the lower bound given in [19]. His method is different from ours.

Remark 1. The algorithm above uses $c\sqrt{dN}$ queries for some constant c and outputs the solution with probability at least $1/2$. In order to reduce the error probability to $1/2^k$ one could run the algorithm k times and, among the dk resulting indices, output the d smallest values of different type. However, starting each run with randomly chosen points of different type regardless of the previous outcome is a waste of information. So it is much more clever to run the algorithm only once and stop it after $kc\sqrt{dN}$ queries.

4. MINIMUM SPANNING TREE. In this section we consider undirected graphs with weighted edges. In MINIMUM SPANNING TREE we wish to compute a cycle-free edge set of maximal cardinality that has minimum total weight. To be precise, if the graph is not connected, this is actually a spanning forest.

Classically, there are a number of different approaches to finding minimum spanning trees efficiently, including the algorithms of Borůvka [5, 20], Kruskal [17], and Prim [21]. To construct an efficient quantum algorithm, we use Borůvka's algorithm since it is of a highly parallel nature. This allows us to use the minima finding algorithms given in section 3.

Borůvka's algorithm consists of at most $\log n$ iterations. In brief, initially it starts with a collection of n spanning trees, each tree containing a single vertex. In each iteration, it finds a minimum weight edge out of each tree in the collection, adds the edges to the trees, and merges them into larger and fewer trees. After at most $\log n$ iterations, there is only one tree left, which is a minimum spanning tree. The correctness of Borůvka's algorithm rests on the following simple fact about spanning trees.

FACT 4.1. *Let $U \subset V$ be a set of vertices of a connected graph $G = (V, E)$ and let e be a minimum weight edge of $(U \times \bar{U}) \cap E$. Then there is a minimum spanning tree containing e .*

In our quantum version of Borůvka's algorithm, we make a few adjustments to keep the overall error probability small without sacrificing the number of queries. We adjust it slightly so that the ℓ th iteration errs with probability at most $\frac{1}{2^{\ell+2}}$, ensuring

that the overall error is at most $\frac{1}{4}$. This increases the cost of the ℓ th iteration by a factor of ℓ , but since the cost of the first few iterations dominates, this is asymptotically negligible. The details follow.

1. Let T_1, T_2, \dots, T_k be a spanning forest. Initially, $k = n$ and each tree T_j contains a single vertex.
2. Set $\ell = 0$.
3. Repeat until there is only a single spanning tree (i.e., $k = 1$).
 - (a) Increment ℓ .
 - (b) Find edges e_1, e_2, \dots, e_k satisfying that e_j is a minimum weight edge leaving T_j . Interrupt when the total number of queries is $(\ell + 2)c\sqrt{km}$ for some appropriate constant c .
 - (c) Add the edges e'_j to the trees, merging them into larger trees.
4. Return the spanning tree T_1 .

To find the minimum edges e_1, \dots, e_k in Step 3(b), we use the following functions. In the array model, any edge (u, v) is coded twice, u appears as neighbor of v , but v also appears as neighbor of u . Enumerate the directed edges from 0 to $2m - 1$. Let $f : [2m] \rightarrow \mathbb{N}^*$ denote the function that maps every directed edge (u, v) to its weight if u and v belong to different trees of the current spanning forest and to ∞ otherwise. Let $g : [2m] \rightarrow [k]$ denote the function that maps every directed edge (u, v) to the index j of the tree T_j containing u . We then apply the algorithm for FIND k SMALLEST VALUES OF DIFFERENT TYPE, interrupting it after $(\ell + 2)c\sqrt{km}$ queries to obtain an error probability at most $1/2^{\ell+2}$ (see Remark 1 in section 3).

THEOREM 4.2. *Given an undirected graph with weighted edges, the algorithm above outputs a spanning tree that is minimum with probability at least $\frac{1}{4}$. The algorithm uses $O(\sqrt{nm})$ queries in the array model and $O(n^{3/2})$ queries in the matrix model.*

Proof. To simplify the proof, consider the matrix model an instance of the array model with $m = n(n - 1)$ edges.

At the beginning of the ℓ th iteration of the main loop, the number of trees k is at most $n/2^{\ell-1}$, and thus it uses at most $(\ell + 2)c\sqrt{nm}/2^{\ell-1}$ queries. Summing over all iterations, the total number of queries is at most $\sum_{\ell \geq 1} (\ell + 2)c\sqrt{nm}/2^{\ell-1}$, which is $O(\sqrt{nm})$.

The ℓ th iteration introduces an error with probability at most $\frac{1}{2^{\ell+2}}$. The overall error probability is thus upper bounded by $\sum_{\ell \geq 1} \frac{1}{2^{\ell+2}} \leq \frac{1}{4}$. \square

5. Connectivity. A special case of MINIMUM SPANNING TREE when all edge weights are equal is GRAPH CONNECTIVITY. The input is an *undirected* graph and the output is a spanning tree, provided the graph is connected.

For the matrix model, the algorithm for MINIMUM SPANNING TREE given in the previous section implies an $O(n^{3/2})$ upper bound for GRAPH CONNECTIVITY as well. Below, we give a somewhat simpler and arguably more natural quantum algorithm of query complexity $O(n^{3/2})$, which is optimal by the lower bound given in section 8 below.

For the array model, we give a quantum algorithm that uses only $O(n)$ queries. Both algorithms start with a collection of n connected components, one for each vertex, and greedily construct a spanning tree by repeatedly picking an edge that connects two of the components.

THEOREM 5.1. *Given the adjacency matrix M of an undirected graph G , the algorithm below outputs a spanning tree for G after an expected number of $O(n^{3/2})$*

queries to M , provided G is connected, and otherwise runs forever.

Proof. Consider the following algorithm.

1. Initially the edge set A is empty.
2. Repeat until A connects the graph.
 - (a) Search for a *good* edge, i.e., an edge that connects two different components in A , and add it to A . Use the version of the search algorithm that returns a solution in the expected $O(n^2/t)$ queries if there are $t > 0$ good edges and otherwise runs forever.
3. Return the edge set A .

Suppose the graph is connected and consider the expected total number of queries used by the algorithm. There are exactly $n-1$ iterations of the main loop. The number of good edges is at least $k-1$ when A consists of k components, and thus the expected total number of queries is on the order of $\sum_{k=2}^n \sqrt{n^2/(k-1)}$, which is $O(n^{3/2})$. \square

When implementing the above algorithm, we maintain an appropriate data structure containing information about the connected components in the graph induced by A . This introduces an additional $O(n \log n)$ term in the running time of the algorithm which is negligible compared to $O(n^{3/2})$. We may choose to stop the algorithm after twice the expected total number of queries, giving an $O(n^{3/2})$ query algorithm with bounded one-sided error.

5.1. The array model. As for the matrix model, we also solve CONNECTIVITY for the array model by constructing larger and larger connected components by merging components. Achieving an $O(n^{3/2})$ query algorithm is fairly easy but not optimal. To do better, we require an efficient routine for generating connected components as a first stage of the complete algorithm.

LEMMA 5.2. *Given an undirected graph G in the array model, we can in $O(n)$ classical queries partition vertex set V into a set of connected components $\{C_1, \dots, C_k\}$ for some integer k , so that for each component C , its total degree $m_C = \sum_{i \in C} d_i$ is no more than $|C|^2$.*

Proof. The algorithm is classical and is as follows.

1. Initially the edge set A is empty.
2. Let $S = V$ be the set of vertices not yet placed in some component.
3. Let $k = 0$ be the number of components constructed thus far.
4. While S is nonempty
 - (a) Take the vertex v of highest degree in S and set $D = \{v\}$.
 - (b) Go through v 's list of neighbors one by one, each time adding the neighbor w to D and the edge (v, w) to A , until one of two events happens: (1) We reach the end of the list, or (2) we reach a neighbor w already assigned to some component C_j with $j \leq k$.
 - (c) In case (1), set $k = k + 1$, $C_k = D$, and remove D from S . In case (2), add D to C_j , and remove D from S .
5. Output k , A , and C_1, C_2, \dots, C_k .

The algorithm uses $n - k$ queries in total, one query for each vertex but the first added to each component. Edge set A contains the union of spanning trees of the components C_1 through C_k .

To show correctness, let v be the vertex chosen in Step 4(a) and d its degree. Then $d \leq |C_j|$ for each component constructed so far, since the size of a freshly created component is the degree of one of its vertices, which by choice in Step 4(a) must be no less than d , and components can only grow. To show that the total degree of every component C_j is no more than $|C_j|^2$, consider the two cases in Step 4(b).

In case (1), D is the set of v and its neighbors, each neighbor having degree no larger than d , implying the total degree is at most $d(d + 1)$ which is strictly less than $(d + 1)^2 = |D|^2$. In case (2), let a be the size of the component C_j to which D is merged, and b the size of D . Then $b \leq d \leq a$. The total degree is no more than $a^2 + bd$ which is strictly less than $(a + b)^2$. \square

THEOREM 5.3. *Given an undirected graph G in the array model, the algorithm below outputs a spanning tree for G using an expected number of $O(n)$ queries, provided G is connected, and otherwise runs forever.*

Proof. Consider the following algorithm.

1. Construct the edge set A using the above lemma.
2. Repeat until A connects the graph.
 - (a) Pick a connected component C in A with smallest total degree, i.e., a component minimizing $m_C = \sum_{i \in C} d_i$.
 - (b) Search for an edge out of C , i.e., an edge that connects C to some other component in A , and add it to A . Use the version of the search algorithm that returns a solution in the expected $O(\sqrt{m_C})$ queries if there is at least one such edge and otherwise runs forever.
3. Return the edge set A .

Suppose the graph is connected and consider the expected total number of queries used by the algorithm.

We first construct k components, each component C having total degree m_C at most $|C|^2$. In each iteration of the main loop, we pick the component with smallest total degree and search for an edge out of C . The expected cost of finding such an edge is at most $\alpha\sqrt{m_C}$ for some constant α . We distribute this cost evenly among each of the m_C edge endpoints in C , each endpoint paying $\alpha/\sqrt{m_C}$.

Fix an arbitrary edge endpoint. Enumerate from 0 up to at most $\log m$ the successive components that were chosen by the algorithm for a search and that contain this fixed edge endpoint. Let m_i be the number of edge endpoints in the i th component. Then $m_{i+1} \geq 2m_i$. The total cost assigned to our fixed edge endpoint is upper bounded by

$$\sum_{i=0}^{\log m} \frac{\alpha}{\sqrt{m_i}} \leq \sum_{i=0}^{\log m} \frac{\alpha}{\sqrt{2^i m_0}} \leq \frac{4\alpha}{\sqrt{m_0}}.$$

Let C be any of the k components constructed in the first step. The total cost assigned over all edge endpoints in C is thus upper bounded by $4\alpha\sqrt{m_C}$, which is at most $4\alpha|C|$. Summing over all k components, the total cost assigned in the main loop is at most $4\alpha n$, which is linear in n . \square

6. STRONG CONNECTIVITY. We give two quantum algorithms for STRONG CONNECTIVITY, first one for the matrix model and then one for the array model. The input is a *directed* graph and the output is a set of at most $2(n - 1)$ edges that proves the graph is strongly connected, provided it is. It follows from the discussions below that such sets always exist.

For the matrix model, all we need is to construct an oriented tree, rooted at some vertex v_0 . We can then run the procedure again on the transposed adjacency matrix, which results in $2(n - 1)$ edges with the required property.

We now give a method for constructing such a tree with bounded error using an optimal number of queries. To keep the overall error small, we classify vertices covered by the current tree into sets T_0, \dots, T_q such that the *confidence* that vertices from T_i

have no new neighbors is increasing with i . Whenever a search for an edge (u, v) with $u \in R$ and $v \notin T_0 \cup \dots \cup T_i$ is successful for some subset $R \subseteq T_i$, the vertices R and v will be moved into T_0 ; otherwise R will be moved into T_{i+1} . We make this formal now.

1. Let S be the tree consisting of the single vertex v_0 .
Partition the vertex set covered by S into $T_0 = \{v_0\}$ and $T_1 = \dots = T_q = \{\}$ for $q = \lfloor \log_2(n) \rfloor + 1$.
2. While there is a set T_i with $|T_i| \geq 2^i$ do
 - (a) Let i be the smallest index such that $|T_i| \geq 2^i$.
 - (b) If $|T_i| < 2^{i+1}$, then $R = T_i$; otherwise R is an arbitrary subset of T_i with $|R| = 2^i$.
 - (c) Remove R from T_i .
 - (d) Search for an edge (u, v) with $u \in R$ and $v \notin S$ in a search space of size $O(2^i n)$ with the version of the quantum search procedure that uses $O(2^{3i/4} \sqrt{n})$ queries and finds a solution with probability $1 - c_0/2^{\sqrt{2^{i+2}}}$, provided such an edge exists, and where $c_0 > 0$ is some appropriately small constant.
 - (e) If the search is successful, add (u, v) to S and $R \cup \{v\}$ to T_0 ; otherwise add R to T_{i+1} .
3. Output S .

We now show some properties of the algorithm. For convenience we define $t_j = |T_0| + |T_1| + \dots + |T_j|$ for each $j = 0, \dots, q$.

LEMMA 6.1. *At the beginning and end of each iteration, the following invariants hold. Let k be the smallest and ℓ the largest index of a nonempty set T_j . Then*

$$(6.1) \quad |T_\ell| \geq 2^{\ell-1},$$

$$(6.2) \quad \forall k \leq j < \ell : t_j \geq 2^j.$$

Proof. The proof is by induction on the number of iterations of the algorithm. Initially, when T_0 is the unique nonempty set, the conditions (6.1) and (6.2) hold.

Assume both conditions hold at the beginning of an iteration. First observe that by the induction hypothesis (6.2) that if $k < \ell$, then $|T_k| \geq 2^k$, and so the index chosen by the algorithm is always $i = k$. Now if the search is successful, we have $t_0 \geq 2^k$ and thus $t_j \geq 2^k$ for all $0 \leq j \leq k$, while for all $j > k$, t_j increases by one, maintaining condition (6.2). If the search is not successful, then by the choice of R , after the iteration, either $t_k = 0$ or $t_k \geq 2^k$, while for all values $j > k$, t_j is not modified, and thus condition (6.2) holds for all $k \leq j < \ell$.

Consider condition (6.1). Whenever an empty set T_j becomes nonempty, it contains at least 2^{j-1} elements, and whenever elements are taken out of a set T_j , afterward it is either empty ($|T_j| = 0$) or $|T_j| \geq 2^j$. Thus in all cases, condition (6.1) is maintained. \square

As a consequence, when the algorithm stops, there is a unique nonempty set T_i and moreover $2^{i-1} \leq |T_i| < 2^i$. Also since at most $n - 1$ searches can be successful, the algorithm stops after $O(n^2)$ iterations.

LEMMA 6.2. *When the algorithm stops, S covers all vertices reachable from v_0 with probability at least $39/40$.*

Proof. First note that the algorithm never outputs a vertex that is not reachable from v_0 . Suppose the algorithm outputs a strict subset of the r vertices that are reachable from v_0 . Consider the probability of the event that the algorithm outputs a subset of size q with $q < r$. For this event to happen, in particular the very last run

of the search procedure fails in finding a new edge, which happens with probability at most $c_0/2^{\sqrt{2^i+2}}$, where i is such that $2^{i-1} \leq q < 2^i$. By summing over all $1 \leq q < r$, the probability that the algorithm fails in finding all reachable vertices is at most

$$\sum_{q \geq 1} \frac{c_0}{2^{\sqrt{q+2}}},$$

which is upper bounded by $\frac{1}{40}$ for some appropriately small constant $c_0 > 0$. \square

We now analyze the complexity of the algorithm.

LEMMA 6.3. *The expected number of queries made by the algorithm is $O(|S|\sqrt{n})$.*

Proof. To analyze the total number of queries made by the search procedures, we group the calls to the search procedures into sequences of unsuccessful searches ending with a success, plus the last sequence of unsuccessful searches.

For the first case, let (u, v) be an arbitrary edge found by the algorithm. Then the probability that it was found when $u \in T_i$ is upper bounded by the probability that it was not found when $u \in T_{i-1}$, which is at most $1/2^{\sqrt{2^i}}$. The cost of this search and the $i - 1$ unsuccessful searches over sets R containing u is of order

$$\sum_{j=0}^i 2^{3j/4} \sqrt{n} \in O(2^{3i/4} \sqrt{n}).$$

The expected cost of finding (u, v) is thus at most

$$\sum_{i \geq 0} \frac{2^{3i/4}}{2^{\sqrt{2^i}}} \sqrt{n} \in O(\sqrt{n}).$$

To complete the analysis we upper bound the total work of the $O(\log n)$ unsuccessful searches which were made after the last successful search. Let i be such that $2^{i-1} \leq |S| < 2^i$. There are at most $2^i/2^j$ searches for sets R with $|R| = 2^j$. Therefore the total work is of order

$$\sum_{j=0}^i 2^{i-j} 2^{3j/4} \sqrt{n} \in O(|S|\sqrt{n}).$$

This concludes the proof. \square

In conclusion we have an algorithm that outputs a directed tree T rooted at v_0 . With probability at least $39/40$, T covers all the vertices which are reachable from v_0 , and its expected number of queries to M is at most $O(n^{3/2})$. This can be turned into an algorithm with worst case query complexity $O(n^{3/2})$ and success probability at least $19/20$. Now running the algorithm again on the transposed adjacency matrix provides an edge set $T \cup T'$ which with probability at least $9/10$ is strongly connected, provided the input graph is strongly connected.

THEOREM 6.4. *There is an algorithm that, given the adjacency matrix M of a directed graph G , uses $O(n^{3/2})$ queries to M , outputs “strongly connected” with probability at least $9/10$ if G is strongly connected, and otherwise outputs “not strongly connected.”*

6.1. The array model. In the array model, we may try to apply an algorithm similar to the one given above for the matrix model. Doing so, we would reverse the edge orientations as part of the second stage, which would require m queries and

hence be of the same query complexity as the classical algorithm that initially queries every bit of the input. We now give another algorithm that is sublinear in m and works in two stages. The first stage computes an oriented spanning tree rooted at v_0 as we did for the matrix model, but this time by using a depth-first search algorithm. The second stage then uses desirable properties of a depth-first search tree to achieve an algorithm of complexity sublinear in m .

LEMMA 6.5. *Given the adjacency array of a directed graph G and a vertex v_0 , the algorithm below uses $O(\sqrt{nm \log n})$ queries to M and outputs a directed tree $A \subseteq E$ rooted at v_0 . With probability at least $\frac{9}{10}$, A is a depth-first tree spanning all of G , provided G is strongly connected.*

Proof. Consider the following simple algorithm.

1. Initially the edge set A is empty.
2. Let $S = \{v_0\}$ be a set of reachable vertices and $T = \{v_0\}$ a stack of vertices to be processed.
3. While $T \neq \{\}$ do
 - (a) Let u be the topmost vertex on stack T .
 - (b) Search for a neighbor v of u not in S . Use the version of the search algorithm that uses $O(\sqrt{d_u^+ \log n})$ queries and outputs a solution with probability at least $1 - \frac{1}{20n}$, provided one exists.
 - (c) If (3b) succeeds, add (u, v) to A , add v to S , and push v onto T .
 - (d) Otherwise, remove u from T .
4. Return edge set A .

For any vertex u let b_u^+ be its out-degree in the tree A produced by the algorithm. Then the total number of queries spent in finding the b_u^+ neighbors of u is on the order of $\sum_{t=1}^{b_u^+} \sqrt{d_u^+ / t} \sqrt{\log n}$ which is $O(\sqrt{b_u^+ d_u^+ \log n})$. Summing over all vertices u this gives

$$\sum_{u \in V} \sqrt{b_u^+ d_u^+ \log n} \leq \sqrt{\sum_u b_u^+} \sqrt{\sum_u d_u^+ \log n} \in O(\sqrt{nm \log n}),$$

where the first inequality follows from the Cauchy–Schwarz inequality and the second from the fact that a tree has only $O(n)$ edges. The algorithm spends in addition $O(\sum_u \sqrt{d_u^+ \log n})$ queries for the unsuccessful neighbor searches, but this is dominated by the previous cost.

The overall error probability is upper bounded by $\frac{1}{10}$ since each of the at most $2n - 1$ searches has error at most $\frac{1}{20n}$. \square

We show now, given a depth-first tree A , how to compute the remaining edge set B such that $A \cup B$ is strongly connected, provided G is strongly connected.

THEOREM 6.6. *Given the adjacency array of a directed graph $G = (V, E)$ and a vertex v_0 , there is an algorithm that uses $O(\sqrt{nm \log n})$ queries and outputs an edge set $E' \subseteq E$ of size at most $2n - 2$ covering v_0 . If G is strongly connected, then with probability at least $\frac{3}{4}$, (V, E') is strongly connected.*

Proof. We use the previous algorithm to construct a directed depth-first spanning tree $A \subseteq E$ rooted at v_0 . We label the vertices according to the order in which they are added to the tree A in Lemma 6.5.

Then in the second stage, for every vertex $v_i \in V$, we search for the neighbor v_j with smallest index. The result is a set of backward edges $B \subseteq E$. We claim that the graph $G = (V, E)$ is strongly connected if and only if its subgraph $G' = (V, A \cup B)$ is strongly connected.

Clearly if G' is strongly connected then so is G since $A \cup B \subseteq E$. Therefore to show the converse assume G is strongly connected. For a proof by contradiction let v_i be the vertex with smallest index for which there is no path from v_i to v_0 in G' . By assumption there is a path in G from v_i to v_0 . Let $(v_l, v_{l'})$ be its first edge with $l \geq i$ and $l' < i$. We use the following property of depth-first search.

Let v_i and v_l be two vertices in the graph G with $i \leq l$. If there is a path from v_i to v_l in G then v_l is in the subtree of $G'' = (V, A)$ that is rooted at v_i .

Therefore we can replace in the original path the portion from v_i to v_l by a path using only edges from A . Let $v_{l''}$ be the neighbor of v_l in G with smallest index. Clearly $l'' \leq l' < i$. By the choice of v_i , there exists a path from $v_{l''}$ to v_0 in G' . Together this gives a path from v_i to v_0 in G' , contradicting the assumption and therefore concluding the correctness of the algorithm.

Now we analyze the complexity. During the first stage, set A is computed in time $O(\sqrt{nm} \log n)$. The second stage can be done with $O(\sqrt{nm})$ queries using the minima finding for the mapping from an edge number in $[1, m]$ to the source-target vertex pair. Both stages can be made to succeed with probability at least $7/8$. \square

7. SINGLE SOURCE SHORTEST PATHS. Let G be a directed graph with nonnegative edge weights and let v_0 be a fixed vertex in G . We want to compute a shortest path from v_0 to every vertex $v \in V$. It may happen that the shortest path is not unique. Using, for example, the lexicographical ordering on vertex sequences, we choose to compute a single canonical shortest path. From now on assume that different paths have different lengths. As a result, the union over all vertices v of the shortest paths from v_0 to v is a *shortest path tree*. Let $\nu(u, v)$ be the weight of edge (u, v) and $\delta(v_0, v)$ the shortest path length from v_0 to v .

Classically SINGLE SOURCE SHORTEST PATHS may be solved by Dijkstra's algorithm. It maintains a subtree \mathcal{T} with the "shortest path subtree" invariant: for any vertex $v \in \mathcal{T}$, the shortest path from v_0 to v uses only vertices from \mathcal{T} . An edge (u, v) is called a *border edge* (of \mathcal{T}) if $u \in \mathcal{T}$ and $v \notin \mathcal{T}$, where u is the source vertex and v the target vertex. The *cost* of (u, v) is $\delta(v_0, u) + \nu(u, v)$. Dijkstra's algorithm starts with $\mathcal{T} = \{v_0\}$ and iteratively adds the cheapest border edge to it.

Our improvement lies in the selection of the cheapest border edge. We first give an algorithm for the array model. Setting $m = n^2$ gives the upper bound for the matrix model.

THEOREM 7.1. *The bounded error query complexity of SINGLE SOURCE SHORTEST PATHS in the array model is $O(\sqrt{nm} \log^2 n)$.*

Proof. As in Dijkstra's algorithm, we iteratively construct a tree T such that for every vertex $v \in T$, the shortest path from v_0 to v is in T . We maintain a partitioning of the vertices covered by T and store the partitioning as a sequence of sets. The length of the sequence is denoted by l .

1. Set $T = \{v_0\}$, $l = 1$, $P_1 = \{v_0\}$.
2. Repeat until T covers the graph
 - (a) For P_l compute up to $|P_l|$ cheapest border edges with disjoint target vertices. For this purpose set $N = \sum_{v \in P_l} d(v)$, and number all edges with source in P_l from 0 to $N - 1$. Define the functions $f : [N] \rightarrow \mathbb{N}^*$ and $g : [N] \rightarrow V$, where $g(i)$ is target vertex of the i th edge and $f(i)$ is its weight if $g(i) \notin T$ and ∞ otherwise. Apply the algorithm of section 3 on f and g with $d = |P_l|$ to find the d edges of smallest weight with distinct target vertices. Let A_l be the resulting edge set.

(b) Let (u, v) be the minimal weighted edge of $A_1 \cup \dots \cup A_l$ with $v \notin P_1 \cup \dots \cup P_l$. Set $T = T \cup \{(u, v)\}$, $P_{l+1} = \{v\}$, and $l = l + 1$.

(c) As long as $l \geq 2$ and $|P_{l-1}| = |P_l|$, merge P_l into P_{l-1} , and set $l = l - 1$.

All steps but 2(a) construct a vertex set sequence P_1, \dots, P_l , the cardinality of each being a power of two, and of strictly decreasing sizes. Figure 7.1 shows an example of this partitioning of the vertices in T .

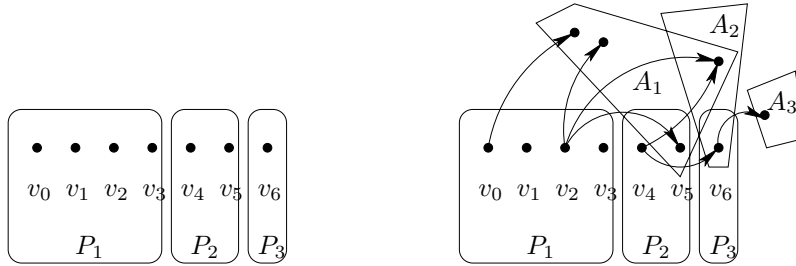


FIG. 7.1. Left: Example of the set decomposition for $|T| = 7$ into powers of 2. Right: Example of corresponding edge sets. The closest border edge of T belongs to one of A_1, A_2, A_3 .

Therefore each set P_i is strictly larger than the union of all the following sets. Hence, if A_i contains $|P_i|$ edges, then at least one of them has its target vertex outside of P_1, \dots, P_l . Let (u, v) be the cheapest border edge of T . Let P_i be the vertex set containing u . Then A_i must contain this edge, and Step 2(b) selects it.

Only Step 2(a) generates queries to the graph. Consider the total number of queries related to sets P_i of some size s . There are at most n/s sets of this size s . Therefore the total work is of order $\sum_{j=1}^{n/s} \sqrt{sm_j}$, where m_j is the number of edges with source in the j th vertex set. We have $\sum_{j=1}^{n/s} m_j = m$. The worst case is when $m_i = sm/n$. In this case, the total work is $O(\sqrt{nm})$ for the fixed size s . There are $\log n$ different set sizes in the algorithm. We require that each of the $O(n \log n)$ queries to the minimum finding succeeds with probability at least $1 - 1/2n \log n$, which introduces an $O(\log n)$ factor (see also Remark 1 in section 3) and we obtain the claimed complexity. \square

8. The lower bounds. Finding the d smallest values of a function can be done in $O(\sqrt{dN})$ queries by Theorem 3.4. We now show this is tight.

THEOREM 8.1. *The problems FIND d SMALLEST VALUES OF A FUNCTION, FIND d ELEMENTS OF DIFFERENT TYPE, and FIND d SMALLEST VALUES OF DIFFERENT TYPE require $\Omega(\sqrt{dN})$ queries.*

Proof. For even k and odd d we consider $d \times k$ boolean matrices with a single 0 in every row. It is encoded by a function $f : [N] \rightarrow \{0, 1\}$ with $N = kd$, such that for every $i \in [d], j \in [k]$, $f(id + j)$ is the entry in row i and column j . Let function $g : [N] \rightarrow [d + 1]$ be such that $g(id + j)$ maps to i if $f(id + j) = 0$ and to d otherwise. By this construction, the problems of FIND $d + 1$ SMALLEST VALUES OF A FUNCTION or $d + 1$ ELEMENTS OF DIFFERENT TYPE or $d + 1$ SMALLEST VALUES OF DIFFERENT TYPE are all equivalent to finding the positions of the d zeros in the matrix.

Let X be the set of matrices such that exactly $\lfloor d/2 \rfloor$ rows have their 0 in the first $k/2$ columns. And let Y be the set of matrices such that this number is exactly $\lceil d/2 \rceil$. We show a lower bound for distinguishing X and Y . We say that matrix $A \in X$ is in relation with $B \in Y$ if and only if the matrices differ at exactly two entries. It follows that there are indices $i \in [d], 0 \leq j < k/2 \leq j' < k$ with $A_{ij} = B_{ij'} = 1$ and

$A_{ij'} = B_{ij} = 0$. The following example illustrates this definition.

$$(8.1) \quad A = \begin{pmatrix} \overbrace{011}^{k/2} \overbrace{111}^{k/2} \\ 111 \ 011 \\ 111 \ 101 \end{pmatrix}, \quad B = \begin{pmatrix} \overbrace{011}^{k/2} \overbrace{111}^{k/2} \\ 110 \ 111 \\ 111 \ 101 \end{pmatrix}.$$

Then the number of matrices which are in relation with a fixed matrix is at least $m = m' = \lceil d/2 \rceil k/2$. For fixed i, j the number of matrices in relation with M and differing at i, j is $k/2$ if $M_{ij} = 0$ and 1 if $M_{ij} = 1$. So $l_{\max} = k/2$ and, by Theorem 2.2, $\Omega(\sqrt{mm'}/l_{\max})$ gives the required lower bound. \square

We first consider lower bounds for CONNECTIVITY and STRONG CONNECTIVITY in the array model. By reducing from PARITY as done by Henzinger and Fredman for the on-line connectivity problem [14], one can show a simple lower bound of $\Omega(n)$ queries in the array model for both problems. For directed graphs, we can improve this to $\Omega(\sqrt{nm})$ queries in the array model.

THEOREM 8.2. STRONG CONNECTIVITY requires $\Omega(\sqrt{nm})$ queries in the array model.

Proof. Let m be such that $m = kn$ for some integer k , and assume $n - k$ is even. We construct the lower bound for regular graphs with out-degree k .

We use two kinds of vertices. The first $2p$ vertices are connected by two disjoint cycles or by a single cycle, as in the aforementioned reduction from PARITY. The other k vertices are used as a pool to collect most of the edges. See Figure 8.1. Let the vertex set be $V = \{v_0, \dots, v_{2p-1}, u_0, \dots, u_{k-1}\}$ for integer $p = (n - k)/2$. In the list model, the edges are defined by a function $f : V \times [k] \rightarrow V$. We consider only functions with the following restrictions:

For every $i \in [k]$ we have $f(u_i, 0) = v_0$ and for $j \in \{1, \dots, k - 1\}$ $f(u_i, j) = u_{i+j}$, where addition is modulo k .

For every $i \in [p]$ there exist $j_0, j_1 \in [k]$ and a bit b such that $f(v_{2i}, j_0) = v_{2i+2+b}$ and $f(v_{2i+1}, j_1) = v_{2i+3-b}$, where addition is modulo $2p$ this time. We call these edges the *forward edges*. The *backward edges* are for all $j \in [k]$ $f(v_{2i}, j) = u_j$ whenever $j \neq j_0$ and $f(v_{2i+1}, j) = u_j$ whenever $j \neq j_1$.

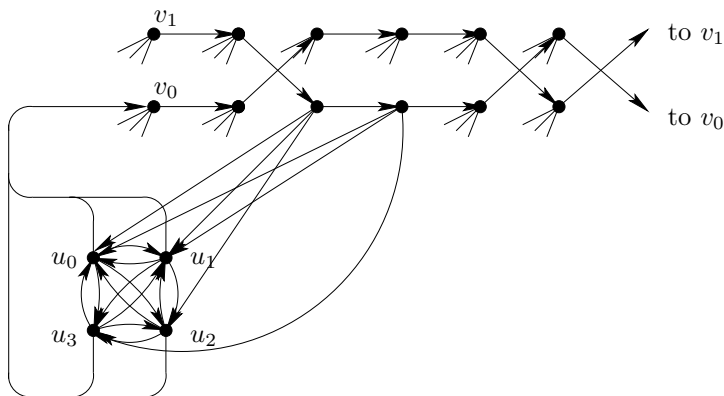


FIG. 8.1. A strongly connected graph.

Now all the vertices are connected to the k -clique, the clique is connected to v_0 , and the graph is strongly connected if and only if there is a path from v_0 to v_1 .

Let X be the set of functions which define a strongly connected graph and Y the set of functions which do not. Function $f \in X$ is in relation with $g \in Y$ if there are numbers $i \in [p], j_0, j_1, h_0, h_1 \in [k]$ with $j_0 \neq h_0, j_1 \neq h_1$ such that the only places where f and g differ are

$$\begin{aligned} g(v_{2i}, h_0) &= f(v_{2i+1}, j_1)g(v_{2i+1}, h_1) = f(v_{2i}, j_0), \\ g(v_{2i}, j_0) &= u_{j_0}g(v_{2i+1}, j_1) = u_{j_1}, \\ f(v_{2i}, h_0) &= u_{h_0}f(v_{2i+1}, h_1) = u_{h_1}. \end{aligned}$$

Informally f and g are in relation if there is a level, where the forward edges are exchanged between a *parallel* and *crossing* configuration and in addition the edge labels are changed.

Then $m = m' \in \Omega(nk^2), p \in \Omega(n)$, for the number of levels and $(k - 1)^2$ for the number of possible forward edge labels. We also have $l_{f,(v,j)} = k - 1$ if $f(v, j) \in \{u_0, \dots, u_{k-1}\}$ and $l_{f,(v,j)} = (k - 1)^2$ otherwise. The value $l'_{g,(v,j)}$ is the same. Since only one of $f(v, j), g(v, j)$ can be in $\{u_0, \dots, u_{k-1}\}$ we have $l_{\max} \in O(k^3)$ and the lower bound follows. \square

For the matrix model, there is a much simpler lower bound which works even for undirected graphs.

THEOREM 8.3. *CONNECTIVITY requires $\Omega(n^{3/2})$ queries in the matrix model.*

Proof. We use Ambainis’s method for the following special problem in a very simple manner. We are given a symmetric matrix $M \in \{0, 1\}^{n \times n}$ with the promise that it is the adjacency matrix of a graph with exactly one or two cycles, and we have to determine which is the case.

Let X be the set of all adjacency matrices of a unique cycle and Y the set of all adjacency matrices with exactly two cycles each of length between $n/3$ and $2n/3$. We define the relation $R \subseteq X \times Y$ as $M R M'$ if there exist $a, b, c, d \in [n]$ such that the only difference between M and M' is that $(a, b), (c, d)$ are edges in M but not in M' and $(a, c), (b, d)$ are edges in M' but not in M . See Figure 8.2. The definition of Y implies that in M the distance from a to c is between $n/3$ and $2n/3$.

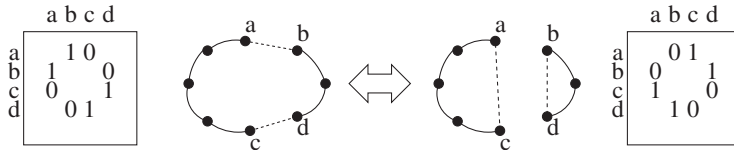


FIG. 8.2. *Illustration of the relation.*

Then $m \in \Omega(n^2)$ since there are $n - 1$ choices for the first edge and $n/3$ choices for the second edge. Also $m' \in \Omega(n^2)$ since from each cycle one edge must be picked, and cycle length is at least $n/3$.

We have $l_{M,(i,j)} = 4$ if $M_{i,j} = 0$ since in M' we have the additional edge (i, j) and the endpoints of the second edge must be neighbors of i and j , respectively. Moreover $l_{M,(i,j)} \in O(n)$ if $M_{i,j} = 1$ since then (i, j) is one of the edges to be removed and there remain $n/3$ choices for the second edge.

The values $l'_{M',(i,j)}$ are similar, so in the product one factor will always be constant while the other is linear giving $l_{M,(i,j)}l'_{M',(i,j)} \in O(n)$, and the theorem follows. \square

We give a lower bound for both **MINIMUM SPANNING TREE** and **SINGLE SOURCE SHORTEST PATHS**.

THEOREM 8.4. MINIMUM SPANNING TREE *and* SINGLE SOURCE SHORTEST PATHS *each require* $\Omega(\sqrt{nm})$ *queries.*

Proof. The proof is a reduction from minima finding. Let $m = k(n + 1)$ for some integer k . Let M be a matrix with n rows and k columns and positive entries. The lower bound on minima finding, with $d = n, N = kn$, shows that $\Omega(\sqrt{kn^2})$ queries are required to find the minimum value in every row.

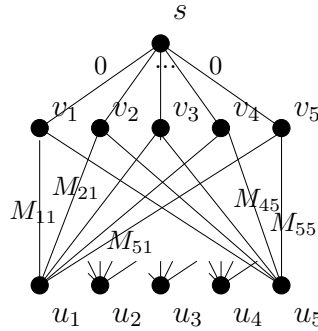


FIG. 8.3. Reduction from finding minima to MINIMAL SPANNING TREE.

We construct a weighted graph G from M like this: The vertices are $V(G) = \{s, v_1, \dots, v_k, u_1, \dots, u_n\}$. The edges are all (s, v_i) with weight 0 and all (v_i, u_j) with weight M_{ji} . (See Figure 8.3.) Then clearly a minimum spanning tree contains the 0-weight edges connecting s to all vertices v_i . And every vertex u_j will be connected to the rest of the graph only with the minimal weighted edge. \square

Acknowledgments. We are grateful to Miklos Santha, Katalin Friedl, Oded Regev, Ronald de Wolf, and Andris Ambainis for helpful discussions or comments.

REFERENCES

- [1] A. AHARONOV, A. AMBAINIS, J. KEMPE, AND U. VAZIRANI, *Quantum walks on graphs*, in Proceedings of the 33rd Annual ACM Symposium on Theory of Computing (STOC), ACM, New York, 2001, pp. 50–59.
- [2] A. AMBAINIS, *Quantum lower bounds by quantum arguments*, J. Comput. System Sci., 64 (2002), pp. 750–767.
- [3] C. H. BENNETT, E. BERNSTEIN, G. BRASSARD, AND U. VAZIRANI, *Strengths and weaknesses of quantum computing*, SIAM J. Comput., 26 (1997), pp. 1510–1523.
- [4] A. BERZINA, A. DUBROVSKY, R. FREIVALDS, L. LACE, AND O. SCEGULNAJA, *Quantum query complexity for some graph problems*, in Proceedings of the 30th Annual Conference on Current Trends in Theory and Practice of Computer Science (SOFSEM), Merin, Czech Republic, 2004, pp. 140–150.
- [5] O. BORŮVKA, *O jistém problému minimálním*, Práce Mor. Přírodově d. spol. v Brně (Acta Societ. Scient. Natur. Moraviae), 3 (1926), pp. 37–58.
- [6] M. BOYER, G. BRASSARD, P. HØYER, AND A. TAPP, *Tight bounds on quantum searching*, Fortschr. Phys., 46 (1998), pp. 493–505.
- [7] G. BRASSARD AND P. HØYER, *An exact quantum polynomial-time algorithm for Simon’s problem*, in Proceedings of the 5th Israeli Symposium on Theory of Computing and Systems (ISTCS), Ramat-Gan, Israel, 1997, pp. 12–23.
- [8] G. BRASSARD, P. HØYER, M. MOSCA, AND A. TAPP, *Quantum amplitude amplification and estimation*, in Quantum Computation and Quantum Information: A Millennium Volume, Contemp. Math. 305, AMS, Providence, RI, 2002, pp. 53–74.
- [9] H. BUHRMAN, R. CLEVE, R. DE WOLF, AND CH. ZALKA, *Bounds for small-error and zero-error quantum algorithms*, in Proceedings of the 40th IEEE Symposium on Foundations of Computer Science (FOCS), IEEE Computer Society, Los Alamitos, CA, 1999, pp. 358–368.

- [10] A. M. CHILDS, R. CLEVE, E. DEOTTO, E. FARHI, S. GUTMANN, AND D. A. SPIELMAN, *Exponential algorithmic speedup by quantum walk*, in Proceedings of the 35th Annual ACM Symposium on Theory of Computing (STOC), ACM, New York, 2003, pp. 59–68.
- [11] CH. DÜRR AND P. HØYER, *A quantum algorithm for finding the minimum*, quant-ph/9607014, 1996.
- [12] E. FARHI, J. GOLDSTONE, S. GUTMANN, AND M. SIPSER, *A limit on the speed of quantum computation in determining parity*, quant-ph/9802045, 1998.
- [13] L. GROVER, *A fast mechanical algorithm for database search*, in Proceedings of the 28th Annual ACM Symposium on Theory of Computing (STOC), ACM, New York, 1996, pp. 212–219.
- [14] M. R. HENZINGER AND M. L. FREDMAN, *Lower bounds for fully dynamic connectivity problems in graphs*, Algorithmica, 22 (1998), pp. 351–362.
- [15] J. KEMPE, *Quantum random walks—an introductory overview*, Contemporary Physics, 44 (2003), pp. 307–327.
- [16] D. KOZEN, *The Design and Analysis of Algorithms*, Springer-Verlag, New York, 1991.
- [17] J. B. KRUSKAL, JR., *On the shortest spanning subtree of a graph and the traveling salesman problem*, in Proc. Amer. Math. Soc., 7 (1956), pp. 48–50.
- [18] A. NAYAK, *Lower Bounds for Quantum Computation and Communication*, Ph.D. thesis, The University of California, Berkeley, CA, 1999.
- [19] A. NAYAK AND F. WU, *The quantum query complexity of approximating the median and related statistics*, in Proceedings of the 31st Annual ACM Symposium on Theory of Computing (STOC), ACM, New York, 1999, pp. 384–393.
- [20] J. NEŠETŘIL, E. MILKOVÁ, AND H. NEŠETŘILOVÁ, *Otakar Borůvka on minimum spanning tree problem: Translation of both the 1926 papers, comments, history*, Discrete Math., 233 (2001), pp. 3–36.
- [21] R. PRIM, *Shortest connecting networks and some generalizations*, Bell System Technical Journal, 36 (1957), pp. 1389–1401.
- [22] J. WATROUS, *Quantum simulations of classical random walks and undirected graph connectivity*, J. Comput. System Sci., 62 (2001), pp. 376–391.

THE APPROXIMABILITY OF THREE-VALUED MAX CSP*

PETER JONSSON[†], MIKAEL KLASSON[†], AND ANDREI KROKHIN[‡]

Abstract. In the maximum constraint satisfaction problem (MAX CSP), one is given a finite collection of (possibly weighted) constraints on overlapping sets of variables, and the goal is to assign values from a given domain to the variables so as to maximize the number (or the total weight, for the weighted case) of satisfied constraints. This problem is **NP**-hard in general, and, therefore, it is natural to study how restricting the allowed types of constraints affects the approximability of the problem. It is known that every Boolean (that is, two-valued) MAX CSP with a finite set of allowed constraint types is either solvable exactly in polynomial time or else **APX**-complete (and hence can have no polynomial-time approximation scheme unless **P** = **NP**). It has been an open problem for several years whether this result can be extended to non-Boolean MAX CSP, which is much more difficult to analyze than the Boolean case. In this paper, we make the first step in this direction by establishing this result for MAX CSP over a three-element domain. Moreover, we present a simple description of all polynomial-time solvable cases of our problem. This description uses the well-known algebraic combinatorial property of supermodularity. We also show that every hard three-valued MAX CSP contains, in a certain specified sense, one of the two basic hard MAX CSPs which are the MAXIMUM k -COLORABLE SUBGRAPH problems for $k = 2, 3$.

Key words. maximum constraint satisfaction, approximability, dichotomy, supermodularity

AMS subject classifications. 68Q25, 90C27

DOI. 10.1137/S009753970444644X

1. Introduction and related work. Many combinatorial optimization problems are **NP**-hard, and the use of approximation algorithms is one of the most prolific techniques to deal with **NP**-hardness. However, hard optimization problems exhibit different behavior with respect to approximability, and complexity theory for approximation is now a well-developed area [1].

Constraint satisfaction problems (CSPs) have always played a central role in this direction of research, since the CSP framework contains many natural computational problems, for example, from graph theory and propositional logic. Moreover, certain CSPs were used to build foundations for the theory of complexity for optimization problems [21], and some CSPs provided material for the first optimal inapproximability results [16] (see also survey [25]). In a CSP, informally speaking, one is given a finite collection of constraints on overlapping sets of variables, and the goal is to decide whether there is an assignment of values from a given domain to the variables satisfying all constraints (decision problem) or to find an assignment satisfying the maximum number of constraints (optimization problem). In this paper we will focus on the optimization problems, which are known as *maximum constraint satisfaction* problems (MAX CSPs). The most well-known examples of such problems are MAX

*Received by the editors November 1, 2004; accepted for publication (in revised form) October 21, 2005; published electronically March 24, 2006.

<http://www.siam.org/journals/sicomp/35-6/44644.html>

[†]Department of Computer and Information Science, University of Linköping, Linköping, S-581 83, Sweden (peter.jonsson@ida.liu.se, mikael.klasson@ida.liu.se). The work of the first author was partially supported by the Swedish Research Council (VR) under grant 621–2003–3421 and by the Center for Industrial Information Technology (CENIIT) under grant 04.01. The work of the second author was partially supported by the Swedish Research Council (VR) under grant 621–2003–3421.

[‡]Corresponding author. Department of Computer Science, University of Durham, Science Laboratories, South Road, Durham, DH1 3LE, UK (andrei.krokhin@durham.ac.uk). The work of this third author was partially supported by the UK EPSRC grant GR/T05325/01.

k -SAT and MAX CUT. Let us now formally define these problems.

Let D denote a *finite* set with $|D| > 1$. Let $R_D^{(m)}$ denote the set of all m -ary predicates over D , that is, functions from D^m to $\{0, 1\}$, and let $R_D = \bigcup_{m=1}^{\infty} R_D^{(m)}$. Also, let \mathbb{Z}^+ denote the set of all nonnegative integers.

DEFINITION 1.1. *A constraint over a set of variables $V = \{x_1, x_2, \dots, x_n\}$ is an expression of the form $f(\mathbf{x})$ where*

- $f \in R_D^{(m)}$ is called the constraint predicate and
- $\mathbf{x} = (x_{i_1}, \dots, x_{i_m})$ is called the constraint scope.

The constraint f is said to be satisfied on a tuple $\mathbf{a} = (a_{i_1}, \dots, a_{i_m}) \in D^m$ if $f(\mathbf{a}) = 1$.

DEFINITION 1.2. *For a finite $\mathcal{F} \subseteq R_D$, an instance of MAX CSP(\mathcal{F}) is a pair (V, C) where*

- $V = \{x_1, \dots, x_n\}$ is a set of variables taking their values from the set D and
- C is a collection of constraints $f_1(\mathbf{x}_1), \dots, f_q(\mathbf{x}_q)$ over V , where $f_i \in \mathcal{F}$ for all $1 \leq i \leq q$.

The goal is to find an assignment $\varphi : V \rightarrow D$ that maximizes the number of satisfied constraints, that is, to maximize the function $f : D^n \rightarrow \mathbb{Z}^+$, defined by $f(x_1, \dots, x_n) = \sum_{i=1}^q f_i(\mathbf{x}_i)$. If the constraints have (positive integral) weights ϱ_i , $1 \leq i \leq q$, then the goal is to maximize the total weight of satisfied constraints, to maximize the function $f : D^n \rightarrow \mathbb{Z}^+$, defined by $f(x_1, \dots, x_n) = \sum_{i=1}^q \varrho_i \cdot f_i(\mathbf{x}_i)$.

Note that throughout the paper the values 0 and 1 taken by any predicate will be considered, rather unusually, as integers, not as Boolean values, and addition will always denote the addition of integers. It is easy to check that, in the Boolean case, our problem coincides with the MAX CSP considered in [9, 10, 19]. We say that a predicate is nontrivial if it is not identically 0. Throughout the paper, we assume that \mathcal{F} is finite and contains only nontrivial predicates.

Boolean constraint satisfaction problems (that is, when $D = \{0, 1\}$) are by far better studied [10] than the non-Boolean version. The main reason for this is, in our opinion, that Boolean constraints can be conveniently described by propositional formulas which provide a flexible and easily manageable tool, and which have been extensively used in complexity theory from its very birth. Moreover, Boolean CSPs suffice to represent a number of well-known problems and to obtain results clarifying the structure of complexity for large classes of interesting problems [10]. In particular, Boolean CSPs were used to provide evidence for one of the most interesting phenomena in complexity theory, namely, that interesting problems belong to a small number of complexity classes [10], which cannot be taken for granted due to Ladner's theorem. After the celebrated work of Schaefer [22] presenting a tractable versus NP-complete dichotomy for Boolean decision CSPs, many classification results have been obtained (see, e.g., [10]), most of which are dichotomies. In particular, a dichotomy in complexity and approximability for the Boolean MAX CSP has been obtained by Creignou [9], and it was slightly refined in [19] (see also [10]).

Many papers on various versions of Boolean CSPs mention studying non-Boolean CSPs as a possible direction of future research, and additional motivation for it, with an extensive discussion, was given by Feder and Vardi [14]. Non-Boolean CSPs provide a much wider variety of computational problems. Moreover, research in non-Boolean CSPs leads to new sophisticated algorithms (e.g., [3]) or to new applications of known algorithms (e.g., [7]). Dichotomy results on non-Boolean CSPs give a better understanding of what makes a computational problem tractable or hard, and they give a more clear picture of the structure of complexity of problems, since many facts

observed in Boolean CSPs appear to be special cases of more general phenomena. Notably, many appropriate tools for studying non-Boolean CSPs have not been discovered until recently. For example, universal algebra tools have proved to be very fruitful when working with decision and counting problems [2, 4, 5, 8] while ideas from combinatorial optimization and operational research have been recently suggested for optimization problems [7].

The MAX-CSP framework has been well studied in the Boolean case. Many fundamental results have been obtained, concerning both complexity classifications and approximation properties (see, e.g., [9, 10, 16, 18, 19, 28]). In the non-Boolean case, a number of results have been obtained that concern exact (superpolynomial) algorithms or approximation properties (see, e.g., [11, 12, 13, 17, 23]). The main research problem we will look at in this paper is the following.

PROBLEM 1. *Classify the problems $\text{MAX CSP}(\mathcal{F})$ with respect to approximability.*

It is known that, for any \mathcal{F} , $\text{MAX CSP}(\mathcal{F})$ is an **NP** optimization (**NPO**) problem that belongs to the complexity class **APX**. In other words, for any \mathcal{F} , there is a polynomial-time approximation algorithm for $\text{MAX CSP}(\mathcal{F})$ whose performance is bounded by a constant.

For the Boolean case, Problem 1 was solved in [9, 10, 19]. It appears that a Boolean $\text{MAX CSP}(\mathcal{F})$ also exhibits a dichotomy in that it either is solvable exactly in polynomial time or else does not admit a polynomial-time approximation scheme (PTAS) unless $\mathbf{P}=\mathbf{NP}$. These papers also describe the boundary between the two cases.

In this paper we solve the above problem for the case $|D| = 3$ by showing that $\text{MAX CSP}(\mathcal{F})$ is solvable exactly in polynomial time if, after removing redundant values, if there are any, from the domain (that is, taking the core), all predicates in \mathcal{F} are supermodular with respect to some linear ordering of the reduced domain (see definitions in section 2.2) or else the problem is **APX**-complete. Experience shows that non-Boolean constraint problems are much more difficult to classify, and hence we believe that the techniques used in this paper can be further extended to all finite domains D . A small technical difference between our result and that of [19] is that we allow repetitions of variables in constraints, as in [10]. Similarly to [10, 19], weights do not play much of a role, since the tractability part of our result holds for the weighted case, while the hardness part is true in the unweighted case even if repetitions of constraints in instances are disallowed. Our result uses a combinatorial property of supermodularity which is a well-known source of tractable optimization problems [6, 15, 24], and the technique of strict implementations [10, 19] which allows one to show that an infinite family of problems can express, in a regular way, one of a few basic hard problems. We remark that the idea to use supermodularity in the analysis of the complexity of $\text{MAX CSP}(\mathcal{F})$ is very new and has not been even suggested in the literature prior to [7]. Generally, it has been known for a while that the property of supermodularity allows one to solve many maximization problems in polynomial time [6, 15, 24]; however, our result is surprising in that supermodularity appears to be the *only* source of tractability for $\text{MAX CSP}(\mathcal{F})$. In the area of approximability, examples of other works where hardness results are obtained for large families of problems simultaneously include [20, 27].

The only other known complete dichotomy result on a non-Boolean constraint problem (that is, with *no* restrictions on \mathcal{F}) is the theorem of Bulatov [2], where the complexity of the standard decision problem CSP on a three-element domain is

classified. Despite the clear similarity in the settings and also in the outcomes (full dichotomy in both cases), we note that none of the universal-algebraic techniques used in [2] can possibly be applied in the study of MAX CSP because the main algebraic constructions which preserve the complexity of decision problems can be easily shown not to do this in the case of optimization problems. Another similarity between Bulatov's result and our theorem is that the proof is broken down to a (relatively) large number of cases. We believe that this is caused either by insufficiently general methods or, more likely, by significant variation in structure of the problems under consideration, where a large number of cases is probably an unavoidable feature of complete classifications.

The structure of the paper is as follows. Section 2 contains definitions of approximation complexity classes and reductions, descriptions of our reduction techniques, and the basics of supermodularity. Section 3 contains the proof of the main theorem of the paper. Finally, section 4 contains a discussion of the work we have done and of possible future work.

2. Preliminaries. This section is subdivided into two parts. The first one contains basic definitions on complexity of approximation and our reduction techniques, while the second one introduces the notion of supermodularity and discusses the relevance of this notion in the study of MAX CSP.

2.1. Approximability.

2.1.1. Definitions. A *combinatorial optimization problem* is defined over a set of *instances* (admissible input data); each instance \mathcal{I} has a finite set $\text{sol}(\mathcal{I})$ of *feasible solutions* associated with it. An *objective* function attributes an integer value to any feasible solution. The *goal* of an optimization problem is, given an instance \mathcal{I} , to find a solution $s \in \text{sol}(\mathcal{I})$ of *optimum* value. The optimal value is the largest one for *maximization* problems and the smallest one for *minimization* problems. A combinatorial optimization problem is said to be an **NP** optimization (**NPO**) problem if instances and solutions can be recognized in polynomial time, solutions are polynomial-bounded in input size, and the objective function can be computed in polynomial time. For a more formal definition, see, e.g., [1].

DEFINITION 2.1 (performance ratio). A solution s to an instance \mathcal{I} of an **NPO** problem Π is r -approximate if it has value Val satisfying

$$\max \left\{ \frac{Val}{Opt(\mathcal{I})}, \frac{Opt(\mathcal{I})}{Val} \right\} \leq r,$$

where $Opt(\mathcal{I})$ is the optimal value for a solution to \mathcal{I} . An approximation algorithm for an **NPO** problem Π has performance ratio $\mathcal{R}(n)$ if, given any instance \mathcal{I} of Π with $|\mathcal{I}| = n$, it outputs an $\mathcal{R}(n)$ -approximate solution.

DEFINITION 2.2 (complexity classes). **PO** is the class of **NPO** problems that can be solved (to optimality) in polynomial time. An **NPO** problem Π is in the class **APX** if there is a polynomial-time approximation algorithm for Π whose performance ratio is bounded by a constant.

Completeness in **APX** is defined using an appropriate reduction, called *AP*-reduction. Our definition of this reduction follows [10, 19].

DEFINITION 2.3 (*AP*-reduction, **APX**-completeness). An **NPO** problem Π_1 is said to be *AP*-reducible to an **NPO** problem Π_2 if two polynomial-time computable functions F and G and a constant α exist such that

1. for any instance \mathcal{I} of Π_1 , $F(\mathcal{I})$ is an instance of Π_2 ;

2. for any instance \mathcal{I} of Π_1 , and any feasible solution s' of $F(\mathcal{I})$, $G(\mathcal{I}, s')$ is a feasible solution of \mathcal{I} ;
3. for any instance \mathcal{I} of Π_1 , and any $r \geq 1$, if s' is an r -approximate solution of $F(\mathcal{I})$, then $G(\mathcal{I}, s')$ is a $(1 + (r - 1)\alpha + o(1))$ -approximate solution of \mathcal{I} where the o -notation is with respect to $|\mathcal{I}|$.

An **NPO** problem Π is **APX**-hard if every problem in **APX** is **AP**-reducible to it. If, in addition, Π is in **APX** then Π is called **APX**-complete.

It is a well-known fact (see, e.g., [1, section 8.2.1]) that **AP**-reductions compose. It is well known that $\text{MAX CSP}(\mathcal{F})$ belongs to **APX** for every \mathcal{F} (actually, it belongs to **MAX SNP** which is a subclass of **APX** [21]); a direct proof of this fact can be found in [7]. A complete classification of the complexity of $\text{MAX CSP}(\mathcal{F})$ for a two-element set D was obtained in [9, 19]; we will give it in subsection 2.2. We shall now give an example of an **APX**-complete problem which will be used extensively in this paper.

Example 2.4. Given a graph $G = (V, E)$, the **MAXIMUM k -COLORABLE SUBGRAPH** problem, $k \geq 2$, is the problem of maximizing $|E'|$, $E' \subseteq E$, such that the graph $G' = (V, E')$ is k -colorable. This problem is known to be **APX**-complete (it is Problem GT33 in [1]). Let neq_k denote the binary disequality predicate on $\{0, 1, \dots, k-1\}$, $k \geq 2$, that is, $neq_k(x, y) = 1 \Leftrightarrow x \neq y$. The problem $\text{MAX CSP}(\{neq_k\})$ is slightly more general than the **MAXIMUM k -COLORABLE SUBGRAPH** problem. To see this, think of vertices of a given graph as of variables, and apply the predicate to every pair of variables x, y such that (x, y) is an edge in the graph.

If we allow weights on edges in graphs and on constraints, then the problems coincide; the obtained problem is known as **MAX k -CUT**. For unweighted problems, the $\text{MAX CSP}(\{neq_k\})$ is slightly more general because, formally, one can have constraints $neq_k(x, y)$ and $neq_k(y, x)$ in the same instance. In any case, it follows that the problem $\text{MAX CSP}(\{neq_k\})$ is **APX**-complete.

Interestingly, the problems $\text{MAX CSP}(\{neq_k\})$, $k = 2, 3$, will be the only basic hard problems for the case $|D| \leq 3$. We will show that, for all other **APX**-complete problems $\text{MAX CSP}(\mathcal{F})$, the set \mathcal{F} can express, in a certain regular approximability-preserving way, one of the predicates neq_2, neq_3 .

2.1.2. Reduction techniques. The basic reduction technique in our **APX**-completeness proofs is based on *strict implementations*; see [10, 19], where this notion was defined and used for the Boolean case. We will give this definition in a slightly different form from that of [10, 19], but it can easily be checked that it is equivalent to the original one.

DEFINITION 2.5. Let $Y = \{y_1, \dots, y_m\}$ and $Z = \{z_1, \dots, z_n\}$ be two disjoint sets of variables. The variables in Y are called *primary* and the variables in Z *auxiliary*. The set Z may be empty. Let $g_1(\mathbf{y}_1), \dots, g_s(\mathbf{y}_s)$, $s > 0$, be constraints over $Y \cup Z$. If $g(y_1, \dots, y_m)$ is a predicate such that the equality

$$g(y_1, \dots, y_m) + (\alpha - 1) = \max_Z \sum_{i=1}^s g_i(\mathbf{y}_i)$$

is satisfied for all y_1, \dots, y_m , and some fixed $\alpha \in \mathbb{Z}^+$, then this equality is said to be a *strict α -implementation of g from g_1, \dots, g_s* .

We use $\alpha - 1$ rather than α in the above equality to ensure that this notion coincides with the original notion of a strict α -implementation for Boolean constraints [10, 19].

We say that a collection of predicates \mathcal{F} *strictly implements* a predicate g if, for some $\alpha \in \mathbb{Z}^+$, there exists a strict α -implementation of g using predicates only from

\mathcal{F} . In this case we write $\mathcal{F} \xrightarrow{s}_\alpha g$. It is not difficult to show that if g can be obtained from \mathcal{F} by a series of strict implementations then it can also be obtained by a single strict implementation. In this paper, we will use about 60 (relatively) short strict implementations for the case when $|D| = 3$. Each of them can be straightforwardly verified by hand, or (better still) by a simple computer program.¹

The following lemma is stated in [10] for the Boolean case, but it is immediately extendable to any finite domain.

LEMMA 2.6 (Lemma 5.18 in [10]). *If \mathcal{F} strictly implements a predicate f , and $\text{MAX CSP}(\mathcal{F} \cup \{f\})$ is **APX**-complete, then $\text{MAX CSP}(\mathcal{F})$ is **APX**-complete as well.*

As in [10], Lemma 2.6 will be the main tool in our **APX**-completeness proofs. It will be used as follows: if \mathcal{F}' is a fixed finite collection of predicates each of which can be strictly implemented by \mathcal{F} , then we can assume that $\mathcal{F}' \subseteq \mathcal{F}$. For example, if \mathcal{F} contains a binary predicate f , then we can assume, at any time when it is convenient, that \mathcal{F} also contains $f'(x, y) = f(y, x)$, since this equality is a strict 1-implementation of f' .

Example 2.7. The (SIMPLE) **MAX CUT** problem is the problem of partitioning the set of vertices of a given undirected graph into two subsets so as to maximize the number of edges with ends being in different subsets. This problem is the same as **MAXIMUM 2-COLORABLE SUBGRAPH** (see Example 2.4), and hence it is **APX**-complete (see Problem ND14 in [1]). As was mentioned in Example 2.4, this problem is essentially the same as $\text{MAX CSP}(\{neq_2\})$. Let f_{dicut} be the binary predicate on $\{0, 1\}$ such that $f_{dicut}(x, y) = 1 \Leftrightarrow x = 0, y = 1$. Then $\text{MAX CSP}(\{f_{dicut}\})$ is essentially the problem **MAX DICUT** (see problem ND16 in [1]), which is the problem of partitioning the vertices of a digraph into two subsets V_0 and V_1 so as to maximize the number of arcs going from V_0 to V_1 . This problem is known to be **APX**-complete as well, and this can be proved by exhibiting a strict 1-implementation from f to neq_2 . Here it is: $neq_2(x, y) = f_{dicut}(x, y) + f_{dicut}(y, x)$.

For a subset $D' \subseteq D$, let $u_{D'}$ denote the predicate such that $u_{D'}(x) = 1$ if and only if $x \in D'$. Let $\mathcal{U}_D = \{u_{D'} \mid \emptyset \neq D' \subseteq D\}$, that is, \mathcal{U}_D is the set of all nontrivial unary predicates on D . We will now give two more examples of strict implementations that will be used later in our proofs.

Example 2.8. Let $D = \{0, 1, 2\}$, and let g_i , $i = 0, 1, 2$, be the binary predicates on D defined by the following rule: $g_i(x, y) = 1 \Leftrightarrow (x = y = i \text{ or } x, y \in D \setminus \{i\})$. We will show that $\mathcal{F} = \{g_0, g_1, g_2\} \cup \mathcal{U}_D$ strictly implements the binary predicate $g(x, y)$ such that $g(x, y) = 1 \Leftrightarrow x = 0, y = 1$. Indeed, one can check that the following is a strict 5-implementation:

$$g(x, y) + 4 = \max_{z, w} [g_0(x, z) + g_1(y, w) + g_2(z, w) + u_{\{0\}}(z) + u_{\{1, 2\}}(w)].$$

Example 2.9. In this example, we will show that the predicate neq_3 can be strictly implemented from the binary equality predicate eq_3 and all unary predicates on $D = \{0, 1, 2\}$. We will use three additional binary predicates f_1, f_2, f_3 defined as follows:

$$\begin{aligned} f_1(x, y) &= 1 \Leftrightarrow x \leq y, \\ f_2(x, y) &= 1 \Leftrightarrow (x, y) = (1, 2), \\ f_3(x, y) &= 1 \Leftrightarrow (x, y) \in \{(1, 0), (1, 2), (2, 0)\}. \end{aligned}$$

¹An example of such a program can be obtained from the authors or anonymously downloaded from <http://www.ida.liu.se/~petej/supermodular.html>.

Then it can be checked that the following equalities hold:

$$\begin{aligned}
 f_1(x, y) + 3 &= \max_{z,w} [eq_3(z, w) + eq_3(z, y) + eq_3(w, x) + u_{\{2\}}(z) + u_{\{1\}}(w) + u_{\{0\}}(x)]; \\
 f_2(x, y) + 5 &= \max_{z,w} [f_1(z, w) + f_1(w, y) + f_1(x, z) + u_{\{0,1\}}(z) + u_{\{0,2\}}(w) + u_{\{1,2\}}(x)]; \\
 f_3(x, y) + 2 &= \max_{z,w} [f_2(z, w) + f_2(z, x) + f_2(w, z) + f_2(w, y) + f_2(x, w) + f_2(x, y) \\
 &\quad + f_2(y, z) + u_{\{0\}}(y)]; \\
 neq_3(x, y) &= f_3(x, y) + f_3(y, x).
 \end{aligned}$$

As mentioned above, any chain of strict implementations can be replaced by a single strict implementation. Since $\text{MAX CSP}(\{neq_3\})$ is **APX**-complete by Example 2.4, this and Lemma 2.6 imply that the problem $\text{MAX CSP}(\{eq_3\} \cup \mathcal{U}_D)$ is **APX**-complete as well. Note that this result was first proved in [7].

Another notion which we will use in our hardness proofs is the notion of a core for a set of predicates. In the case when \mathcal{F} consists of a single binary predicate h , this notion coincides with the usual notion of a core of the directed graph whose arcs are specified by h .

DEFINITION 2.10. *An endomorphism of \mathcal{F} is a unary operation π on D such that we have $f(a_1, \dots, a_m) = 1 \Rightarrow f(\pi(a_1), \dots, \pi(a_m)) = 1$ for all $f \in \mathcal{F}$ and all $(a_1, \dots, a_m) \in D^m$. We will say that \mathcal{F} is a core if every endomorphism of \mathcal{F} is injective (i.e., a permutation).*

If π is an endomorphism of \mathcal{F} with a minimal image $im(\pi) = D'$, then a core of \mathcal{F} , denoted $core(\mathcal{F})$, is the subset $\{f|_{D'} \mid f \in \mathcal{F}\}$ of $R_{D'}$.

The intuition here is that if \mathcal{F} is not a core, then it has a noninjective endomorphism π , which implies that, for every assignment φ , there is another assignment $\pi\varphi$ that satisfies all constraints satisfied by φ and uses only a restricted set of values, so the problem is equivalent to a problem over this smaller set. As in the case of graphs, all cores of \mathcal{F} are isomorphic, so one can speak about *the* core of \mathcal{F} . The following rather simple lemma will be frequently used in our proofs.

LEMMA 2.11. *If $\mathcal{F}' = core(\mathcal{F})$ and $\text{MAX CSP}(\mathcal{F}')$ is **APX**-complete, then so is $\text{MAX CSP}(\mathcal{F})$.*

Proof. We produce an *AP*-reduction from $\text{MAX CSP}(\mathcal{F}')$ to $\text{MAX CSP}(\mathcal{F})$. We may assume that the endomorphism $\pi : D \rightarrow D'$ is the identity on D' , since if it is not, then some power of π is such an endomorphism. We will now describe functions F and G necessary for the reduction. The function F takes an instance of $\text{MAX CSP}(\mathcal{F}')$ and replaces every predicate $f|_{D'}$ in it by f . If \mathcal{I} is an instance of $\text{MAX CSP}(\mathcal{F}')$, with the set V of variables, and s' is a feasible solution of $F(\mathcal{I})$ (that is, an assignment $V \rightarrow D$), then $G(F(\mathcal{I}), s') = s$ defined by $s(x) = \pi(s'(x))$ for all $x \in V$. It is easy to see that s is also a feasible solution for \mathcal{I} . Finally, note that, since π is an endomorphism, s satisfies every constraint satisfied by s' ; in particular, we have $Opt(\mathcal{I}) = Opt(F(\mathcal{I}))$. Hence, if s' is an r -approximate solution for $F(\mathcal{I})$, then s is an r -approximate solution for \mathcal{I} , so we can choose $\alpha = 1$ in the definition of *AP*-reducibility. \square

Example 2.12. Let f be the binary predicate on $\{0, 1\}$ considered in Example 2.7 and g the binary predicate on $\{0, 1, 2\}$ considered in Example 2.8. It is easy to see that $\{f\}$ is the core of $\{g\}$ where the corresponding endomorphism is given by $\pi(0) = 0, \pi(1) = \pi(2) = 1$. Since $\text{MAX CSP}(\{f\})$ is **APX**-complete, Lemma 2.11

implies that $\text{MAX CSP}(\{g\})$ is **APX**-complete as well. Now note that this also proves that the problem $\text{MAX CSP}(\{g_0, g_1, g_2\} \cup \mathcal{U}_{\{0,1,2\}})$, as considered in Example 2.8, is **APX**-complete.

2.2. Supermodularity. In this section we discuss the well-known combinatorial algebraic property of supermodularity [24], which will play a crucial role in classifying the approximability of MAX CSP problems.

A partial order on a set D is called a *lattice order* if, for every $x, y \in D$, there exists a greatest lower bound $x \sqcap y$ and a least upper bound $x \sqcup y$. The corresponding algebra $\mathcal{L} = (D, \sqcap, \sqcup)$ is called a *lattice*. For tuples $\mathbf{a} = (a_1, \dots, a_n)$, $\mathbf{b} = (b_1, \dots, b_n)$ in D^n , let $\mathbf{a} \sqcap \mathbf{b}$ and $\mathbf{a} \sqcup \mathbf{b}$ denote the tuples $(a_1 \sqcap b_1, \dots, a_n \sqcap b_n)$ and $(a_1 \sqcup b_1, \dots, a_n \sqcup b_n)$, respectively.

DEFINITION 2.13. *Let \mathcal{L} be a lattice on D . A function $f : D^n \rightarrow \mathbb{Z}^+$ is called supermodular on \mathcal{L} if*

$$f(\mathbf{a}) + f(\mathbf{b}) \leq f(\mathbf{a} \sqcap \mathbf{b}) + f(\mathbf{a} \sqcup \mathbf{b}) \text{ for all } \mathbf{a}, \mathbf{b} \in D^n,$$

and f is called submodular on \mathcal{L} if the inverse inequality holds.

We say that $\mathcal{F} \subseteq R_D$ is supermodular on \mathcal{L} if every $f \in \mathcal{F}$ has this property.

A finite lattice $\mathcal{L} = (D, \sqcap, \sqcup)$ is *distributive* if and only if it can be represented by subsets of a set A , where the operations \sqcap and \sqcup are interpreted as set-theoretic intersection and union, respectively. Totally ordered lattices, or *chains*, will be of special interest in this paper. Note that, for chains, the operations \sqcap and \sqcup are simply min and max. Hence, the supermodularity property for an n -ary predicate f on a chain is expressed as follows:

$$f(\mathbf{a}) + f(\mathbf{b}) \leq f(\min(a_1, b_1), \dots, \min(a_n, b_n)) + f(\max(a_1, b_1), \dots, \max(a_n, b_n))$$

for all $\mathbf{a} = (a_1, \dots, a_n)$ and $\mathbf{b} = (b_1, \dots, b_n)$.

Example 2.14. (1) The binary equality predicate eq_3 is not supermodular on any chain on $\{0, 1, 2\}$. Take, without loss of generality, the chain $0 < 1 < 2$. Then

$$eq_3(1, 1) + eq_3(0, 2) = 1 \not\leq 0 = eq_3(0, 1) + eq_3(1, 2).$$

(2) Reconsider the predicates neq_2 and f from Example 2.7. It is easy to check that neither of them is supermodular on any chain on $\{0, 1\}$.

(3) Fix a chain on D and let \mathbf{a}, \mathbf{b} be arbitrary elements of D^2 . Consider the binary predicates $f_{\mathbf{a}}$, $f^{\mathbf{b}}$, and $f_{\mathbf{a}}^{\mathbf{b}}$ defined by the rules

$$\begin{aligned} f_{\mathbf{a}}(x, y) &= 1 \Leftrightarrow (x, y) \leq \mathbf{a}, \\ f^{\mathbf{b}}(x, y) &= 1 \Leftrightarrow (x, y) \geq \mathbf{b}, \\ f_{\mathbf{a}}^{\mathbf{b}}(x, y) &= 1 \Leftrightarrow (x, y) \leq \mathbf{a} \text{ or } (x, y) \geq \mathbf{b}, \end{aligned}$$

where the order on D^2 is componentwise. It is easy to check that every predicate of the form $f_{\mathbf{a}}$, $f^{\mathbf{b}}$, or $f_{\mathbf{a}}^{\mathbf{b}}$ is supermodular on the chain. Note that such predicates were considered in [7], where they were called generalized 2-monotone. We will see later in this subsection that such predicates are generic supermodular binary predicates on a chain.

We will now make some very simple, but useful, observations.

h_1	h_2	h_3	h_4	h_5	h_6	h_7	h_8	h_9	h_{10}
$\begin{matrix} 000 \\ 000 \\ 001 \end{matrix}$	$\begin{matrix} 000 \\ 000 \\ 011 \end{matrix}$	$\begin{matrix} 000 \\ 011 \\ 011 \end{matrix}$	$\begin{matrix} 100 \\ 000 \\ 000 \end{matrix}$	$\begin{matrix} 100 \\ 000 \\ 001 \end{matrix}$	$\begin{matrix} 100 \\ 000 \\ 011 \end{matrix}$	$\begin{matrix} 100 \\ 011 \\ 011 \end{matrix}$	$\begin{matrix} 100 \\ 100 \\ 000 \end{matrix}$	$\begin{matrix} 100 \\ 100 \\ 001 \end{matrix}$	$\begin{matrix} 100 \\ 100 \\ 011 \end{matrix}$
$\begin{matrix} 100 & 110 & 110 \\ h_{11} 101 & h_{12} 110 & h_{13} 110 \\ 001 & 000 & 001 \end{matrix}$									

FIG. 2.1. A list of binary predicates on $\{0, 1, 2\}$ that are supermodular on the chain $0 < 1 < 2$. The predicates are represented by matrices, the order of indices being also $0 < 1 < 2$.

OBSERVATION 2.15.

1. Any chain is a distributive lattice.
2. Any lattice on a three-element set is a chain.
3. Any unary predicate on D is supermodular on any chain on D .
4. A predicate is supermodular on a chain if and only if it is supermodular on its dual chain (obtained by reversing the order).

The tractability part of our classification is contained in the following result.

THEOREM 2.16 (see [7]). *If \mathcal{F} is supermodular on some distributive lattice on D , then weighted MAX CSP(\mathcal{F}) is in **PO**.*

Given a binary predicate $f : D^2 \rightarrow \{0, 1\}$, we will often use a $|D| \times |D|$ 0/1-matrix M to represent f : $f(x, y) = 1$ if and only if $M_{xy} = 1$. Note that this matrix is essentially the table of values of the predicate. For example, some binary predicates on $D = \{0, 1, 2\}$ that are supermodular on the chain $0 < 1 < 2$ are listed in Figure 2.1. Matrices for all other binary predicates that are supermodular on $0 < 1 < 2$ can be obtained from those in the list or from the trivial binary predicate by transposing matrices (which corresponds to swapping arguments in a predicate) and by replacing some all-0 rows by all-1 rows, and the same for all-0 columns (but not for both rows and columns at the same time). This can be shown by using Lemma 2.3 [6] or by direct exhaustive (computer-assisted) search. Note that all predicates in Figure 2.1 have the form described in Example 2.14 (3). For example, h_2 is $f^{(2,1)}$ and h_9 is $f^{(2,2)}_{(1,0)}$.

The property of supermodularity can be used to classify the approximability of Boolean problems MAX CSP(\mathcal{F}) (though originally the classification was obtained and stated [9, 10, 19] without using this property). It is easy to see that $\mathcal{F} \subseteq R_{\{0,1\}}$ is not a core if and only if $f(a, \dots, a) = 1$ for some $a \in \{0, 1\}$ and all $f \in \mathcal{F}$, in which case MAX CSP(\mathcal{F}) is trivial.

THEOREM 2.17 (see [7, 10]). *Let $D = \{0, 1\}$ and $\mathcal{F} \subseteq R_D$ be a core. If \mathcal{F} is supermodular on some chain on D , then MAX CSP(\mathcal{F}) belongs to **PO**. Otherwise, MAX CSP(\mathcal{F}) is **APX**-complete.*

Remark 2.18. It was shown in Lemma 5.37 of [10] that, for $|D| = 2$, \mathcal{F} can strictly implement neg_2 whenever $\mathcal{F} \subseteq R_D$ is a core that is not supermodular on any chain on D .

Combining Theorem 2.17 with Lemma 2.11, we get the following corollary which will be used often in our **APX**-completeness proofs.

COROLLARY 2.19. *If g' is binary predicate on $\{0, 1, 2\}$ and $core(\{g'\})$ is $\{g\}$ where g is nonsupermodular predicate on a two-element subset of D , then MAX CSP($\{g'\}$) is **APX**-complete.*

Note that there are only two (up to swapping of arguments) binary predicates g on $\{0, 1\}$ such that $\{g\}$ is a core: the predicates neg_2 and f_{dicut} from Example 2.7. As mentioned above, these two predicates are nonsupermodular, and f_{dicut} strictly 1-implements neg_2 .

3. Main result. In this section we establish a generalization of Theorem 2.17 to the case of a three-element domain. Throughout this section, let $D = \{0, 1, 2\}$. Note that if $\mathcal{F} \subseteq R_D$ is not a core, then, by Lemma 2.11, the problem $\text{MAX CSP}(\mathcal{F})$ either is trivial (if \mathcal{F} has a constant endomorphism) or else reduces to a similar problem over a two-element domain, in which case Theorem 2.17 applies.

THEOREM 3.1. *Let $D = \{0, 1, 2\}$ and $\mathcal{F} \subseteq R_D$ be a core. If \mathcal{F} is supermodular on some chain on D , then weighted $\text{MAX CSP}(\mathcal{F})$ belongs to **PO**. Otherwise, unweighted $\text{MAX CSP}(\mathcal{F})$ is **APX**-complete even if repetitions of constraints in instances are disallowed.*

Proof. The tractability part of the proof follows immediately from Theorem 2.16 (see also Observation 2.15 (1)). Assume for the rest of this section that \mathcal{F} is a core and it is not supermodular on any chain on D . We will show that (at least) one of neg_2, neg_3 can be obtained from \mathcal{F} by using the following two operations:

1. replacing \mathcal{F} by $\mathcal{F} \cup \{f\}$ where f is a predicate that can be strictly implemented from \mathcal{F} ;
2. taking the core of a subset of \mathcal{F} .

By Example 2.4 and Lemmas 2.6 and 2.19, this will establish the result.

To improve readability, we divide the rest of the proof into three parts: in subsection 3.1, we establish **APX**-completeness for some small sets \mathcal{F} consisting of at most two binary and several unary predicates, and also for the case when \mathcal{F} contains an irreflexive nonunary predicate (see definition below). Subsection 3.2 establishes the result when all unary predicates are available, and subsection 3.3 finishes the proof. \square

Remark 3.2. Note that it can be checked in polynomial time whether a given \mathcal{F} is supermodular on some chain on D if the predicates in \mathcal{F} are given by full tables of values or only by tuples on which predicates take value 1.

3.1. Small cases and irreflexive predicates. We say that an n -ary predicate f on D is *irreflexive* if and only if $f(d, \dots, d) = 0$ for all $d \in D$. It is easy to check that any irreflexive nontrivial predicate f is not supermodular on any chain on D . For example, if f is binary and $f(a, b) = 1$ for some $a \neq b$, then $f(a, b) + f(b, a) \geq 1$, but $f(\min(a, b), \min(b, a)) + f(\max(a, b), \max(b, a)) = 0$ due to irreflexivity.

Since a predicate f is supermodular on a chain C if and only if f is supermodular on its dual, we can identify chains on the three-element set D with the same middle element: let C_i denote an arbitrary chain on D with i as its middle element. We also define the set \mathcal{Q}_i that consists of all binary predicates on D that are supermodular on C_i but on neither of the other two chains. For example, it is easy to check using Figure 2.1 that \mathcal{Q}_1 consists of predicates $h_2, h_5, h_6, h_8, h_9, h_{10}, h_{11}$ and the predicates obtained from them by using the following operations:

1. swapping the variables (this corresponds to transposing the tables);
2. adding a unary predicate $u(x)$ or $u(y)$ in such a way that the sum is also a predicate (this corresponds to replacing some all-0 rows/columns by all-1 rows/columns).

Recall that, for a subset $D' \subseteq D$, $u_{D'}$ denotes the predicate such that $u_{D'}(x) = 1$ if and only if $x \in D'$, and $\mathcal{U}_D = \{u_{D'} \mid \emptyset \neq D' \subseteq D\}$; that is, \mathcal{U}_D is the set of all nontrivial unary predicates on D .

LEMMA 3.3. *Let g be a binary predicate such that, for some $a \in D$, $g(x, a) = 1$ for all $x \in D$. Let $g'(x, y) = 0$ if $y = a$ and $g'(x, y) = g(x, y)$ otherwise. Then, the following hold:*

1. *for any chain on D , g and g' are supermodular (or not) on it simultaneously and*
2. *$\{g, u_{D \setminus \{a\}}\}$ strictly implements g' .*

Proof. The first statement can be straightforwardly verified by using the definition of supermodularity. To see that the second statement holds, we note that $g'(x, y) + 1 = g(x, y) + u_{D \setminus \{a\}}(y)$ is a strict 2-implementation of $g'(x, y)$. \square

We say that a predicate g contains an *all-one column* if there exists $a \in D$ such that $g(x, a) = 1$ for all $x \in D$, and we define *all-one rows* analogously. Clearly, the lemma above holds for both all-one rows and all-one columns. The lemma will be used in our hardness proofs as follows: if \mathcal{F} contains $g(x, y)$ and $u_{D \setminus \{a\}}(y)$, then, by Lemma 2.6, we may also assume that $g' \in \mathcal{F}$.

The following lemma contains more **APX**-completeness results for some problems MAX CSP(\mathcal{F}) where \mathcal{F} is a small set containing at most two binary and some unary predicates.

LEMMA 3.4. *Let f, h be binary predicates on D . The problem MAX CSP(\mathcal{F}) is **APX**-complete if one of the following holds:*

1. $\mathcal{F} = \{f\}$ where f is nontrivial and irreflexive;
2. $\mathcal{F} = \{f\} \cup \mathcal{U}_D$ where f is not supermodular on any chain on D ;
3. $\mathcal{F} = \{f, h_7\} \cup \mathcal{U}_D$ where $f \in \mathcal{Q}_0$ and h_7 is given in Figure 2.1;
4. $\mathcal{F} = \{f, h\} \cup \mathcal{U}_D$ where $f \in \mathcal{Q}_1$ and $h \in \mathcal{Q}_0$;
5. $\mathcal{F} = \{f, u_{\{0,1\}}\}$ where f is such that $f(0, 0) = f(1, 1) = 0$ and $f(2, 2) = f(0, 1) = 1$.

Proof. The lemma is proved by providing strict implementations, from \mathcal{F} , of the predicate neq_3 (see Example 2.4) or of a binary predicate whose core is a nonsupermodular predicate on a two-element subset of D (see Corollary 2.19). In total, we give 54 implementations.

We prove only case 1 here; the other cases are similar and can be found in the appendix. First, we make the list of predicates we need to consider. There are 63 irreflexive nontrivial predicates on D . We may skip all predicates whose cores are nonsupermodular predicates on a two-element subset of D , since we already have the result for them (Corollary 2.19). For every pair of predicates that can be obtained from each other by swapping the variables (that is, $f(x, y)$ and $f'(x, y) = f(y, x)$), we can skip one of them. By symmetry, we may skip any predicate obtained from some predicate already in the list by renaming the elements of D . Finally, we already know that the result is true for the disequality predicate neq_3 , so we skip that one too. All this can be done using a computer or by hand, and the resulting list contains only six predicates. Here are strict implementations for them.

$$\begin{array}{l}
 \begin{array}{ccc}
 & 011 & 011 \\
 1. \ f_1 := & \begin{array}{ccc} 001 & \xrightarrow{s} & 101 \\ 000 & & 110 \end{array} & = \ neq_3 \\
 \text{Implementation:} & \ neq_3 = f_1(x, y) + f_1(y, x) & \\
 & \begin{array}{ccc} 010 & & 011 \\
 2. \ f_2 := & \begin{array}{ccc} 001 & \xrightarrow{s} & 101 \\ 100 & & 110 \end{array} & = \ neq_3 \\
 \text{Implementation:} & \ neq_3(x, y) = f_2(x, y) + f_2(y, x) &
 \end{array}
 \end{array}$$

$$3. f_3 := \begin{array}{ccc} & 011 & 010 \\ & 101 \xrightarrow{s} & 001 = f_2 \\ & 100 & 100 \end{array}$$

Implementation: $f_2(x, y) + 2 = \max_z [f_3(z, x) + f_3(x, y) + f_3(y, z)]$

$$4. f_4 := \begin{array}{ccc} & 011 & 011 \\ & 101 \xrightarrow{s} & 101 = neq_3 \\ & 000 & 110 \end{array}$$

Implementation:

$neq_3(x, y) + 2 = \max_z [f_4(z, x) + f_4(z, y) + f_4(x, y) + f_4(y, x)]$

$$5. f_5 := \begin{array}{ccc} & 011 & 011 \\ & 100 \xrightarrow{s} & 101 = f_4 \\ & 000 & 000 \end{array}$$

Implementation:

$f_4(x, y) + 2 = \max_{z,w} [f_5(z, w) + f_5(z, y) + f_5(w, y) + f_5(x, w) + f_5(y, z)]$

$$6. f_6 := \begin{array}{ccc} & 011 & 011 \\ & 001 \xrightarrow{s} & 101 = f_3 \\ & 100 & 100 \end{array}$$

Implementation:

$f_3(x, y) + 3 = \max_{z,w} [f_6(z, y) + f_6(w, z) + f_6(w, x) + f_6(x, z) + f_6(x, y)] \quad \square$

PROPOSITION 3.5. *If $h \in R_D^{(n)}$, $n \geq 2$, is nontrivial and irreflexive, then $\text{MAX CSP}(\{h\})$ is **APX**-complete.*

Proof. The proof is by induction on n (the arity of h). The basis when $n = 2$ was proved in Lemma 3.4(1). Assume that the lemma holds for $n = k$, $k \geq 2$. We show that it holds for $n = k + 1$. Assume first that there exists $(a_1, \dots, a_{k+1}) \in D^{k+1}$ such that $h(a_1, \dots, a_{k+1}) = 1$ and $|\{a_1, \dots, a_{k+1}\}| \leq k$. We assume without loss of generality that $a_k = a_{k+1}$ and consider the predicate $h'(x_1, \dots, x_k) = h(x_1, \dots, x_k, x_k)$. Note that this is a strict 1-implementation of h' , that $h'(d, \dots, d) = 0$ for all $d \in D$, and that h' is nontrivial since $h'(a_1, \dots, a_k) = 1$. Consequently, $\text{MAX CSP}(\{h'\})$ is **APX**-complete by the induction hypothesis, and $\text{MAX CSP}(\{h\})$ is **APX**-complete, too.

Assume now that $|\{a_1, \dots, a_{k+1}\}| = k + 1$ whenever $h(a_1, \dots, a_{k+1}) = 1$. Consider the predicate $h'(x_1, \dots, x_k) = \max_y h(x_1, \dots, x_k, y)$, and note that this is a strict 1-implementation of h' . We see that $h'(d, \dots, d) = 0$ for all $d \in D$ (due to the condition above) and h' is nontrivial since h is nontrivial. We can once again apply the induction hypothesis and draw the conclusion that $\text{MAX CSP}(\{h'\})$ and $\text{MAX CSP}(\{h\})$ are **APX**-complete. \square

3.2. When all unary predicates are available. As the next step, we will prove that $\text{MAX CSP}(\mathcal{F} \cup \mathcal{U}_D)$ is **APX**-complete if \mathcal{F} is not supermodular on any chain. As a special case of Lemma 6.3 of [6], we have the following result (see also Observation 6.1 of [6]).

LEMMA 3.6. *An n -ary, $n \geq 2$, predicate f is supermodular on a fixed chain C if and only if the following holds: every binary predicate obtained from f by replacing any given $n - 2$ variables by any constants is supermodular on C .*

PROPOSITION 3.7. *The problem $\text{MAX CSP}(\mathcal{F} \cup \mathcal{U}_D)$ is **APX**-complete if \mathcal{F} is not supermodular on any chain.*

Proof. By our initial assumptions, \mathcal{F} is not supermodular on any chain. For $i = 0, 1, 2$, let $f_i \in \mathcal{F}$ be not supermodular on C_i . Recall that every unary predicate is supermodular on any chain. Therefore, f_i is n -ary where $n \geq 2$ (note that n depends on i). By Lemma 3.6, it is possible to substitute constants for some $n - 2$ variables of f_i to obtain a binary predicate f'_i which is not supermodular on C_i . Assume without loss

of generality that these variables are the last $n - 2$ variables, and the corresponding constants are d_3, \dots, d_n ; that is, $f'_i(x, y) = f_i(x, y, d_3, \dots, d_n)$. Then the following is a strict $(n - 1)$ -implementation of f'_i :

$$f'_i(x, y) + (n - 2) = \max_{z_3, \dots, z_n} [f_i(x, y, z_3, \dots, z_n) + u_{\{d_3\}}(z_3) + \dots + u_{\{d_n\}}(z_n)].$$

By Lemma 2.6, it now is sufficient to show the result for $\mathcal{F} = \{f'_0, f'_1, f'_2\}$. We can assume that \mathcal{F} consists of at most three binary predicates and it is minimal with the property of not being supermodular on any chain. In addition, we can assume that the binary predicates in \mathcal{F} do not contain any all-one columns or all-one rows (this is justified by Lemma 3.3). We need to consider three cases depending on the number of predicates in \mathcal{F} .

Case 1. $|\mathcal{F}| = 1$. The result is proved in Lemma 3.4 (2).

Case 2. $|\mathcal{F}| = 2$. Assume $\mathcal{F} = \{g, h\}$. We consider two subcases:

1. g is supermodular on C_1 and C_2 but not on C_0 ; this implies that $h \in \mathcal{Q}_0$ because otherwise $\mathcal{F} \cup \mathcal{U}_D$ is supermodular on C_1 or C_2 , or else h is not supermodular on any chain, contradicting the minimality of \mathcal{F} . By Lemma 3.3, we can assume that neither g nor h have an all-1 row or column. It can be easily checked by inspecting the list of binary predicates (see, e.g., Figure 2.1) that there exist only three such predicates g . These are predicates h_3, h_4 , and h_7 from Figure 2.1. We have that $h_4(x, y) + 1 = h_3(x, y) + u_{\{0\}}(x) + u_{\{0\}}(y)$ is a strict 2-implementation of h_4 from h_3 , $h_3(x, y) + 1 = h_4(x, y) + u_{\{1,2\}}(x) + u_{\{1,2\}}(y)$ is a strict 2-implementation of h_3 from h_4 , and $h_7(x, y) = h_3(x, y) + h_4(x, y)$ is a strict 1-implementation of h_7 . Hence, since all unary predicates are available, it is enough to show the result for $g = h_7$, which has already been obtained in Lemma 3.4(3).
2. None of the predicates g, h is supermodular on two distinct (that is, not mutually dual) chains. By symmetry, we may assume that $g \in \mathcal{Q}_1$ and $h \in \mathcal{Q}_0$. Then the result follows from Lemma 3.4(4).

Case 3. $|\mathcal{F}| = 3$. By the minimality of \mathcal{F} , it follows that $\mathcal{F} = \{g_0, g_1, g_2\}$ where each g_i is *not* supermodular on C_i but is supermodular on the other two chains. As argued in the previous case, we may assume that $g_0 = h_7$. By symmetry, we may assume that g_1 and g_2 have the following matrices, respectively:

$$\begin{matrix} 1 & 0 & 1 & & 1 & 1 & 0 \\ 0 & 1 & 0 & \text{and} & 1 & 1 & 0 \\ 1 & 0 & 1 & & 0 & 0 & 1 \end{matrix}$$

It remains to say that, for such \mathcal{F} , **APX**-completeness of MAX CSP($\mathcal{F} \cup \mathcal{U}_D$) was shown in Examples 2.8 and 2.12. \square

3.3. The last step. We will need one more auxiliary lemma. Let $\mathcal{C}_D = \{u_{\{d\}} \mid d \in D\}$.

LEMMA 3.8. *For any \mathcal{F} , if MAX CSP($\mathcal{F} \cup \mathcal{U}_D$) is **APX**-complete, then so is MAX CSP($\mathcal{F} \cup \mathcal{C}_D$).*

Proof. For any disjoint subsets S, T of D , $u_{S \cup T}(x) = u_S(x) + u_T(x)$ is a strict 1-implementation of $u_{S \cup T}$. Use this repeatedly and apply Lemma 2.6. \square

PROPOSITION 3.9. *Let \mathcal{F} be a core. If \mathcal{F} is not supermodular on any chain, then MAX CSP(\mathcal{F}) is **APX**-complete.*

Proof. If \mathcal{F} contains a nontrivial irreflexive predicate then the result follows from Proposition 3.5. Letting $r(f) = \{d \in D \mid f(d, \dots, d) = 1\}$ for a predicate f , we can now assume that $r(f) \neq \emptyset$ for all $f \in \mathcal{F}$. Let $r(\mathcal{F}) = \{r(f) \mid f \in \mathcal{F}\}$. If $r(f) = S$ then $u_S(x) = f(x, \dots, x)$ is a strict 1-implementation of $u_S(x)$. Hence, for each $S \in r(\mathcal{F})$, we can without loss of generality assume that $u_S \in \mathcal{F}$. Note that if, for some $d \in D$, we have $d \in r(f)$ for all $f \in \mathcal{F}$, then the operation sending all elements of D to d is an endomorphism of \mathcal{F} , contradicting the assumption that \mathcal{F} is a core. Hence, for every $d \in D$, there is a unary predicate $u_S \in \mathcal{F}$ (depending on d) such that $d \notin S$.

Note that if $\{a, b, c\} = D$, then $u_{\{b\}}(x) + 1 = u_{\{a,b\}}(x) + u_{\{b,c\}}(x)$ is a strict 2-implementation of $u_{\{b\}}(x)$. Hence, we may assume that, for any distinct two-element sets S_1, S_2 in $r(\mathcal{F})$, we also have $S_1 \cap S_2 \in r(\mathcal{F})$. It is easy to see that then $r(\mathcal{F})$ contains at least one of the following: (1) two distinct singletons, or (2) sets $\{a, b\}$ and $\{c\}$ such that $\{a, b, c\} = D$. We will consider these two cases separately.

Note that, by Proposition 3.7, $\text{MAX CSP}(\mathcal{F} \cup \mathcal{U}_D)$ is **APX**-complete. Then, by Lemma 3.8, $\text{MAX CSP}(\mathcal{F} \cup \mathcal{C}_D)$ is **APX**-complete as well. Hence, by Lemma 2.6, showing that \mathcal{F} can strictly implement every predicate in \mathcal{C}_D is sufficient to prove the proposition.

Case 1. $u_{\{a\}}, u_{\{b\}} \in \mathcal{F}$ and $a \neq b$. Assume without loss of generality that $a = 0$ and $b = 1$. We will show that \mathcal{F} can strictly implement $u_{\{2\}}$. Since \mathcal{F} is a core, let $f \in \mathcal{F}$ be an n -ary predicate witnessing that the operation π_1 such that $\pi_1(0) = 0$ and $\pi_1(1) = \pi_1(2) = 1$ is not an endomorphism of \mathcal{F} . Let $\mathbf{a} = (a_1, \dots, a_n)$ be a tuple such that $f(\mathbf{a}) = 1$, but $f(\pi_1(\mathbf{a})) = 0$ (where $\pi_1(\mathbf{a}) = (\pi_1(a_1), \dots, \pi_1(a_n))$). Note that at least one of the a_i 's must be equal to 2, since otherwise $\mathbf{a} = \pi_1(\mathbf{a})$. For each $1 \leq i \leq n$, let t_i be x if $a_i = 2$ and z_i otherwise. Denote by l the number of t_i 's that are of the form z_i . Now it is not difficult to verify that

$$g_1(x) + l = \max_{\{z_i \mid a_i \neq 2\}} \left[f(t_1, \dots, t_n) + \sum_{a_i \neq 2} u_{\{a_i\}}(z_i) \right]$$

is a strict $(l + 1)$ -implementation of a unary predicate $g_1(x)$ such that $g_1(2) = 1$ and $g_1(1) = 0$. That is, g_1 is either $u_{\{2\}}$ or $u_{\{0,2\}}$. If $g_1 = u_{\{2\}}$ then we have all predicates from \mathcal{C}_D , and we are done. So assume that $g_1 = u_{\{0,2\}}$.

Next, the operation π_2 such that $\pi_2(1) = 1$ and $\pi_2(0) = \pi_2(2) = 0$ is not an endomorphism of \mathcal{F} either. Acting as above, one can show that \mathcal{F} strictly implements a unary predicate $g_2(x)$ such that $g_2(2) = 1$ and $g_2(0) = 0$, which is either $u_{\{2\}}(x)$ or $u_{\{1,2\}}(x)$. Again, if $g_2 = u_{\{2\}}$ then we are done. Otherwise, $g_2 = u_{\{1,2\}}$ and $u_{\{2\}}(x) + 1 = g_1(x) + g_2(x)$ is strict 2-implementation of $u_{\{2\}}$.

Case 2. $u_{\{a,b\}}, u_{\{c\}} \in \mathcal{F}$ and $\{a, b, c\} = D$. Assume without loss of generality that $a = 0, b = 1$, and $c = 2$. Let $f \in \mathcal{F}$ be a predicate witnessing that the operation π such that $\pi(0) = \pi(1) = 1$ and $\pi(2) = 2$ is not an endomorphism of \mathcal{F} . If f is unary then $f = u_{\{0\}}$ or $f = u_{\{0,2\}}$. In the former case we go back to Case 1, and in the latter case $u_{\{0\}}(x) + 1 = u_{\{0,1\}}(x) + u_{\{0,2\}}(x)$ is a strict 2-implementation of $u_{\{0\}}$, so we can use Case 1 again.

Assume that f is n -ary, $n \geq 2$. Similarly to Case 1, let $\mathbf{a} = (a_1, \dots, a_n)$ be a tuple such that $f(\mathbf{a}) = 1$, but $f(\pi(\mathbf{a})) = 0$. For each $1 \leq i \leq n$, let t_i be x if $a_i = 0, y$ if $a_i = 1$, and z otherwise. Note that y or/and z may not appear among the t_i 's (unlike x which does appear). We consider the case when z does appear; the other case is very similar. If none of the t_i 's is y then $g_1(x) + 1 = \max_z [f(t_1, \dots, t_n) + u_{\{2\}}(z)]$ is a strict 2-implementation of a unary predicate g_1 which is either $u_{\{0\}}$ or $u_{\{0,2\}}$ (since

π is not its endomorphism). Hence we are done, as above. Assume now that some t_i is y . Now it is not difficult to verify that $g_2(x, y) + 1 = \max_z [f(t_1, \dots, t_n) + u_{\{2\}}(z)]$ is a strict 2-implementation of a binary predicate g_2 which satisfies $g_2(0, 1) = 1$ and $g_2(1, 1) = 0$. If $g_2(0, 0) = 1$, then the predicate $g_2(x, x)$ is either $u_{\{0\}}$ or $u_{\{0,2\}}$, and we are done. Otherwise, we have $g_2(0, 0) = 0$. Now apply Lemma 3.4 (1) if $g_2(2, 2) = 0$, and use Lemma 3.4 (5) otherwise. \square

4. Conclusion. We have proved a dichotomy result for maximum constraint satisfaction problems over a three-element domain. The property of supermodularity appears to be the dividing line: those sets of predicates whose cores have this property give rise to problems solvable exactly in polynomial time, while all other sets of predicates can implement, in a regular way, the disequality predicate on a two- or three-element set, and hence give rise to **APX**-complete problems. Interestingly, the description of polynomial cases is based on orderings of the domain, which is not suggested in any way by the formulation of the problem.

It can be shown using Theorem 2.16 that Theorem 3.1, as stated in the paper, does not hold for domains with at least four elements. The reason is that all lattices on at most three-element set are chains, but on larger sets there are other types of lattices (for example, a Boolean lattice on a four-element set). For $|D| \geq 4$, Theorem 2.16 can be used to construct examples of sets \mathcal{F} such that $\text{MAX CSP}(\mathcal{F})$ is tractable, and \mathcal{F} is supermodular on some distributive lattice which is not a chain, but not supermodular on any chain. Hence, more general lattices are required to make further progress in classifying the complexity of MAX CSPs, as is a better understanding of the supermodularity property on arbitrary lattices. We believe that the ideas from this paper can be further developed to obtain a complete classification of approximability of MAX CSP.

Notably, the hard problems of the form considered in this paper do not have a PTAS. It is possible that, as in Theorem 8.8 of [10], a restriction on the incidence graph of variables in instance can give rise to **NP**-hard problems that do have a PTAS.

Finally, techniques of [26] can probably be used to obtain better implementations and more precise (in)approximability results for non-Boolean problems MAX CSP. We leave this direction for future research.

Appendix. Proof of Lemma 3.4 (Cases 2–5).

In each case, we generate a list of all applicable predicates, and then optimize it as follows:

- skip a predicate $f(x, y)$ if $f'(x, y)$ is in the list, with $f(x, y) = f'(y, x)$;
- skip all predicates with an all-1 row or column.

By Lemma 3.3, it is sufficient to prove the result for the optimized lists.

Cases 2–5 follow in order, with a description and a list of strict implementations. Each strict implementation produces neq_3 or some binary predicate whose core is a nonsupermodular predicate on a two-element subset of D , or some other predicate for which an implementation has already been found.

Cases 2 and 3 will also use eq_3 —the binary equality predicate—as an implementation target. Note that it was shown in Example 2.9 that the set $\{eq_3\} \cup \mathcal{U}_D$ strictly implement neq_3 , and hence, $\text{MAX CSP}(\{eq_3\} \cup \mathcal{U}_D)$ is **APX**-complete.

If some implementation produces a predicate g , whose core is a nonsupermodular predicate on a two-element subset of D , then we write $([0, 1, 2] \mapsto [\pi(0), \pi(1), \pi(2)])$ to describe the endomorphism π leading to the core, and we also give a matrix for that nonsupermodular predicate.

In all strict implementations in this section, the variables x, y are primary, and z, w (when they appear) are auxiliary.

Case 2. $\mathcal{F} = \{f\} \cup \mathcal{U}_D$ and f is not supermodular on any chain on D . We further optimize the list of predicates for this case. We can assume that $f(d, d) = 1$ for some d , as the other predicates are handled in Case 1. By symmetry, we may skip a predicate if there is another predicate in the list by renaming the elements of D . We can also skip eq_3 , as we have handled it separately in Example 2.9.

$$\left\{ \begin{array}{l} f_1 := \\ 110 \\ 010 \\ 000 \end{array} \right\} \cup \mathcal{U}_D \xrightarrow{s} \begin{array}{l} 100 \\ 010 \\ 001 \end{array} =: g \quad g = eq_3$$

$$\text{Implementation: } g(x, y) + 1 = f_1(x, y) + f_1(y, x) + u_{\{2\}}(x) + u_{\{2\}}(y)$$

$$\left\{ \begin{array}{l} f_2 := \\ 101 \\ 101 \\ 000 \end{array} \right\} \cup \mathcal{U}_D \xrightarrow{s} \begin{array}{l} 000 \\ 000 \\ 010 \end{array} =: g \quad g \text{ has core } \begin{array}{l} 00 \\ 10 \end{array} ([0, 1, 2] \mapsto [1, 1, 2])$$

$$\text{Implementation: } g(x, y) + 1 = f_2(x, y) + u_{\{2\}}(x) + u_{\{1\}}(y)$$

$$\left\{ \begin{array}{l} f_3 := \\ 110 \\ 001 \\ 100 \end{array} \right\} \cup \mathcal{U}_D \xrightarrow{s} \begin{array}{l} 000 \\ 001 \\ 010 \end{array} =: g \quad g \text{ has core } \begin{array}{l} 01 \\ 10 \end{array} ([0, 1, 2] \mapsto [1, 1, 2])$$

$$\text{Implementation: } g(x, y) + 2 = f_3(x, y) + f_3(y, x) + u_{\{1,2\}}(x) + u_{\{1,2\}}(y)$$

$$\left\{ \begin{array}{l} f_4 := \\ 110 \\ 011 \\ 101 \end{array} \right\} \cup \mathcal{U}_D \xrightarrow{s} \begin{array}{l} 100 \\ 010 \\ 001 \end{array} =: g \quad g = eq_3$$

$$\text{Implementation: } g(x, y) + 1 = f_4(x, y) + f_4(y, x)$$

$$\left\{ \begin{array}{l} f_5 := \\ 101 \\ 100 \\ 000 \end{array} \right\} \cup \mathcal{U}_D \xrightarrow{s} \begin{array}{l} 000 \\ 101 \\ 000 \end{array} =: g \quad g \text{ has core } \begin{array}{l} 00 \\ 10 \end{array} ([0, 1, 2] \mapsto [0, 1, 0])$$

$$\text{Implementation: } g(x, y) + 2 = \max_z [f_5(z, y) + f_5(x, z) + u_{\{2\}}(z) + u_{\{1,2\}}(x)]$$

$$\left\{ \begin{array}{l} f_6 := \\ 001 \\ 010 \\ 000 \end{array} \right\} \cup \mathcal{U}_D \xrightarrow{s} \begin{array}{l} 110 \\ 010 \\ 000 \end{array} =: g \quad g = f_1$$

$$\text{Implementation: } g(x, y) + 1 = \max_z [f_6(z, x) + f_6(y, z) + u_{\{0\}}(x)]$$

$$\left\{ \begin{array}{l} f_7 := \\ 011 \\ 010 \\ 000 \end{array} \right\} \cup \mathcal{U}_D \xrightarrow{s} \begin{array}{l} 110 \\ 010 \\ 000 \end{array} =: g \quad g = f_1$$

$$\text{Implementation: } g(x, y) + 2 = \max_z [f_7(z, x) + f_7(z, y) + f_7(y, z) + u_{\{2\}}(z) + u_{\{0\}}(x)]$$

$$\left\{ \begin{array}{l} f_8 := \\ 001 \\ 110 \\ 000 \end{array} \right\} \cup \mathcal{U}_D \xrightarrow{s} \begin{array}{l} 001 \\ 000 \\ 000 \end{array} =: g \quad g \text{ has core } \begin{array}{l} 01 \\ 00 \end{array} ([0, 1, 2] \mapsto [0, 0, 2])$$

$$\text{Implementation: } g(x, y) + 3 = \max_z [f_8(z, y) + f_8(x, z) + u_{\{1,2\}}(z) + u_{\{0,2\}}(x) + u_{\{2\}}(y)]$$

$$\left\{ \begin{array}{l} f_9 := \\ 101 \\ 110 \\ 000 \end{array} \right\} \cup \mathcal{U}_D \xrightarrow{s} \begin{array}{l} 001 \\ 000 \\ 000 \end{array} =: g \quad g \text{ has core } \begin{array}{l} 01 \\ 00 \end{array} ([0, 1, 2] \mapsto [0, 0, 2])$$

$$\text{Implementation: } g(x, y) + 3 = \max_z [f_9(z, y) + f_9(x, z) + u_{\{1\}}(z) + u_{\{0,2\}}(x) + u_{\{2\}}(y)]$$

$$\left\{ \begin{array}{l} f_{10} := \\ 011 \\ 110 \\ 000 \end{array} \right\} \cup \mathcal{U}_D \xrightarrow{s} \begin{array}{l} 001 \\ 010 \\ 000 \end{array} =: g \quad g = f_6$$

Implementation:

$$g(x, y) + 3 = \max_z [f_{10}(x, z) + f_{10}(x, y) + f_{10}(y, z) + u_{\{0,2\}}(z) + u_{\{2\}}(x) + u_{\{2\}}(y)]$$

$$\left\{ \begin{array}{l} f_{11} := \\ \end{array} \begin{array}{l} 001 \\ 010 \\ 100 \end{array} \right\} \cup \mathcal{U}_D \xrightarrow{s} \begin{array}{l} 100 \\ 010 \\ 001 \end{array} =: g \quad g = eq_3$$

Implementation: $g(x, y) + 1 = \max_z [f_{11}(z, x) + f_{11}(z, y)]$

$$\left\{ \begin{array}{l} f_{12} := \\ \end{array} \begin{array}{l} 011 \\ 010 \\ 100 \end{array} \right\} \cup \mathcal{U}_D \xrightarrow{s} \begin{array}{l} 110 \\ 010 \\ 000 \end{array} =: g \quad g = f_1$$

Implementation: $g(x, y) + 2 = \max_z [f_{12}(z, y) + f_{12}(x, z) + u_{\{1,2\}}(z)]$

$$\left\{ \begin{array}{l} f_{13} := \\ \end{array} \begin{array}{l} 011 \\ 110 \\ 100 \end{array} \right\} \cup \mathcal{U}_D \xrightarrow{s} \begin{array}{l} 011 \\ 010 \\ 000 \end{array} =: g \quad g = f_7$$

Implementation: $g(x, y) + 3 = \max_z [f_{13}(z, x) + f_{13}(z, y) + f_{13}(x, y) + u_{\{0\}}(z) + u_{\{0\}}(x)]$

$$\left\{ \begin{array}{l} f_{14} := \\ \end{array} \begin{array}{l} 110 \\ 011 \\ 100 \end{array} \right\} \cup \mathcal{U}_D \xrightarrow{s} \begin{array}{l} 000 \\ 000 \\ 110 \end{array} =: g \quad g \text{ has core } \begin{array}{l} 00 \\ 10 \end{array} \quad ([0, 1, 2] \mapsto [0, 0, 2])$$

Implementation: $g(x, y) + 2 = \max_z [f_{14}(z, y) + f_{14}(x, z) + u_{\{2\}}(x)]$

$$\left\{ \begin{array}{l} f_{15} := \\ \end{array} \begin{array}{l} 101 \\ 011 \\ 100 \end{array} \right\} \cup \mathcal{U}_D \xrightarrow{s} \begin{array}{l} 110 \\ 010 \\ 000 \end{array} =: g \quad g = f_1$$

Implementation: $g(x, y) + 3 = \max_z [f_{15}(z, x) + f_{15}(z, y) + f_{15}(x, z) + u_{\{2\}}(z) + u_{\{1\}}(y)]$

$$\left\{ \begin{array}{l} f_{16} := \\ \end{array} \begin{array}{l} 011 \\ 011 \\ 100 \end{array} \right\} \cup \mathcal{U}_D \xrightarrow{s} \begin{array}{l} 000 \\ 000 \\ 100 \end{array} =: g \quad g \text{ has core } \begin{array}{l} 00 \\ 10 \end{array} \quad ([0, 1, 2] \mapsto [0, 0, 2])$$

Implementation: $g(x, y) + 3 = \max_z [f_{16}(z, y) + f_{16}(x, z) + f_{16}(x, y) + u_{\{2\}}(z) + u_{\{2\}}(x)]$

$$\left\{ \begin{array}{l} f_{17} := \\ \end{array} \begin{array}{l} 110 \\ 010 \\ 001 \end{array} \right\} \cup \mathcal{U}_D \xrightarrow{s} \begin{array}{l} 110 \\ 010 \\ 000 \end{array} =: g \quad g = f_1$$

Implementation: $g(x, y) + 2 = \max_z [f_{17}(z, y) + f_{17}(x, z) + u_{\{0,1\}}(z)]$

$$\left\{ \begin{array}{l} f_{18} := \\ \end{array} \begin{array}{l} 101 \\ 110 \\ 001 \end{array} \right\} \cup \mathcal{U}_D \xrightarrow{s} \begin{array}{l} 110 \\ 010 \\ 000 \end{array} =: g \quad g = f_1$$

Implementation: $g(x, y) + 2 = \max_z [f_{18}(z, x) + f_{18}(y, z) + u_{\{0,1\}}(x)]$

$$\left\{ \begin{array}{l} f_{19} := \\ \end{array} \begin{array}{l} 110 \\ 100 \\ 000 \end{array} \right\} \cup \mathcal{U}_D \xrightarrow{s} \begin{array}{l} 000 \\ 101 \\ 000 \end{array} =: g \quad g \text{ has core } \begin{array}{l} 00 \\ 10 \end{array} \quad ([0, 1, 2] \mapsto [0, 1, 0])$$

Implementation:

$g(x, y) + 4 = \max_{z,w} [f_{19}(z, w) + f_{19}(z, y) + f_{19}(w, x) + u_{\{1\}}(z) + u_{\{1\}}(w) + u_{\{1,2\}}(x) + u_{\{2\}}(y)]$

$$\left\{ \begin{array}{l} f_{20} := \\ \end{array} \begin{array}{l} 101 \\ 010 \\ 100 \end{array} \right\} \cup \mathcal{U}_D \xrightarrow{s} \begin{array}{l} 001 \\ 010 \\ 000 \end{array} =: g \quad g = f_6$$

Implementation:

$g(x, y) + 4 = \max_{z,w} [f_{20}(z, w) + f_{20}(z, y) + f_{20}(w, x) + u_{\{2\}}(z) + u_{\{1,2\}}(w) + u_{\{1,2\}}(y)]$

$$\left\{ \begin{array}{l} f_{21} := \\ \end{array} \begin{array}{l} 101 \\ 011 \\ 110 \end{array} \right\} \cup \mathcal{U}_D \xrightarrow{s} \begin{array}{l} 101 \\ 010 \\ 100 \end{array} =: g \quad g = f_{20}$$

Implementation: $g(x, y) + 3 = \max_z [f_{21}(z, x) + f_{21}(z, y) + f_{21}(x, y) + u_{\{0,2\}}(z)]$

Case 3. $\mathcal{F} = \{f, h_7\} \cup \mathcal{U}_D$ and $f \in \mathcal{Q}_0$. We further optimize the list of predicates for this case as follows. By symmetry, we can skip a predicate if there is another predicate in the list obtained by swapping the names of elements 1 and 2 of D .

$$\left\{ \begin{array}{l} f_1 := 000 \\ 000 \\ 101 \end{array} \right\} \cup \mathcal{U}_D \xrightarrow{s} \begin{array}{l} 100 \\ 010 \\ 001 \end{array} =: g \quad g = eq_3$$

Implementation: $g(x, y) + 2 = f_1(x, y) + f_1(y, x) + h_7(x, y) + u_{\{0,1\}}(x) + u_{\{0,1\}}(y)$

$$\left\{ \begin{array}{l} f_2 := 000 \\ 110 \\ 101 \end{array} \right\} \cup \mathcal{U}_D \xrightarrow{s} \begin{array}{l} 100 \\ 010 \\ 001 \end{array} =: g \quad g = eq_3$$

Implementation: $g(x, y) + 1 = f_2(x, y) + h_7(x, y) + u_{\{0\}}(y)$

$$\left\{ \begin{array}{l} f_3 := 000 \\ 010 \\ 001 \end{array} \right\} \cup \mathcal{U}_D \xrightarrow{s} \begin{array}{l} 000 \\ 110 \\ 101 \end{array} =: g \quad g = f_2$$

Implementation: $g(x, y) + 1 = \max_z [f_3(z, x) + f_3(z, y) + u_{\{0\}}(y)]$

$$\left\{ \begin{array}{l} f_4 := 001 \\ 110 \\ 001 \end{array} \right\} \cup \mathcal{U}_D \xrightarrow{s} \begin{array}{l} 000 \\ 000 \\ 101 \end{array} =: g \quad g = f_1$$

Implementation: $g(x, y) + 2 = \max_z [f_4(z, x) + f_4(y, z) + u_{\{2\}}(z)]$

$$\left\{ \begin{array}{l} f_5 := 000 \\ 010 \\ 101 \end{array} \right\} \cup \mathcal{U}_D \xrightarrow{s} \begin{array}{l} 000 \\ 110 \\ 101 \end{array} =: g \quad g = f_2$$

Implementation: $g(x, y) + 1 = \max_z [f_5(x, z) + f_5(y, z) + u_{\{0\}}(y)]$

Case 4. $\mathcal{F} = \{f, h\} \cup \mathcal{U}_D$ and $f \in \mathcal{Q}_1$ and $h \in \mathcal{Q}_0$. We show that $\{f\} \cup \mathcal{U}_D$ can strictly implement h_7 , whereby the result follows from Case 3.

$$\left\{ \begin{array}{l} f_1 := 100 \\ 000 \\ 011 \end{array} \right\} \cup \mathcal{U}_D \xrightarrow{s} \begin{array}{l} 100 \\ 011 \\ 011 \end{array} =: g \quad g = h_7$$

Implementation: $g(x, y) + 1 = \max_z [f_1(z, x) + f_1(z, y)]$

$$\left\{ \begin{array}{l} f_2 := 100 \\ 100 \\ 011 \end{array} \right\} \cup \mathcal{U}_D \xrightarrow{s} \begin{array}{l} 100 \\ 011 \\ 011 \end{array} =: g \quad g = h_7$$

Implementation: $g(x, y) + 1 = \max_z [f_2(z, x) + f_2(z, y)]$

$$\left\{ \begin{array}{l} f_3 := 100 \\ 100 \\ 000 \end{array} \right\} \cup \mathcal{U}_D \xrightarrow{s} \begin{array}{l} 100 \\ 000 \\ 011 \end{array} =: g \quad g = f_1$$

Implementation:
 $g(x, y) + 2 = \max_z [f_3(z, x) + f_3(z, y) + f_3(x, y) + u_{\{2\}}(z) + u_{\{2\}}(x) + u_{\{1,2\}}(y)]$

$$\left\{ \begin{array}{l} f_4 := 100 \\ 100 \\ 001 \end{array} \right\} \cup \mathcal{U}_D \xrightarrow{s} \begin{array}{l} 100 \\ 100 \\ 000 \end{array} =: g \quad g = f_3$$

Implementation: $g(x, y) + 3 = \max_z [f_4(z, y) + f_4(x, z) + u_{\{2\}}(z) + u_{\{0,1\}}(x) + u_{\{0,1\}}(y)]$

$$\left\{ \begin{array}{l} f_5 := 100 \\ 101 \\ 001 \end{array} \right\} \cup \mathcal{U}_D \xrightarrow{s} \begin{array}{l} 100 \\ 100 \\ 001 \end{array} =: g \quad g = f_4$$

Implementation: $g(x, y) + 3 = \max_z [f_5(z, x) + f_5(z, y) + f_5(x, z) + u_{\{0\}}(z) + u_{\{1,2\}}(x)]$

$$\left\{ \begin{array}{l} f_6 := 000 \\ 000 \\ 011 \end{array} \right\} \cup \mathcal{U}_D \xrightarrow{s} \begin{array}{l} 100 \\ 100 \\ 000 \end{array} =: g \quad g = f_3$$

Implementation: $g(x, y) + 1 = f_6(x, y) + u_{\{0,1\}}(x) + u_{\{0\}}(y)$

$$\left\{ \begin{array}{l} f_7 := 100 \\ 000 \\ 001 \end{array} \right\} \cup \mathcal{U}_D \xrightarrow{s} \begin{array}{l} 100 \\ 101 \\ 001 \end{array} =: g \quad g = f_5$$

Implementation: $g(x, y) + 1 = \max_z [f_7(z, x) + f_7(z, y) + u_{\{1\}}(x)]$

Case 5. $\mathcal{F} = \{f, u_{\{0,1\}}\}$ where f is such that $f(0, 0) = f(1, 1) = 0$ and $f(2, 2) = f(0, 1) = 1$.

$$\left\{ \begin{array}{l} 010 \\ f_1 := 001, u_{\{0,1\}} \\ 101 \end{array} \right\} \xrightarrow{s} \begin{array}{l} 010 \\ 100 \\ 000 \end{array} =: g \quad g \text{ has core } \begin{array}{l} 01 \\ 10 \end{array} ([0, 1, 2] \mapsto [0, 1, 0])$$

$$\text{Implementation: } g(x, y) + 2 = f_1(x, y) + f_1(y, x) + u_{\{0,1\}}(x) + u_{\{0,1\}}(y)$$

$$\left\{ \begin{array}{l} 011 \\ f_2 := 000, u_{\{0,1\}} \\ 011 \end{array} \right\} \xrightarrow{s} \begin{array}{l} 010 \\ 100 \\ 000 \end{array} =: g \quad g \text{ has core } \begin{array}{l} 01 \\ 10 \end{array} ([0, 1, 2] \mapsto [0, 1, 0])$$

$$\text{Implementation: } g(x, y) + 2 = f_2(x, y) + f_2(y, x) + u_{\{0,1\}}(x) + u_{\{0,1\}}(y)$$

$$\left\{ \begin{array}{l} 010 \\ f_3 := 101 \\ 101 \end{array} \right\} \xrightarrow{s} \begin{array}{l} 010 \\ 001 \\ 101 \end{array} =: g \quad g = f_1$$

$$\text{Implementation: } g(x, y) + 2 = \max_z [f_3(z, x) + f_3(x, y) + f_3(y, z)]$$

$$\left\{ \begin{array}{l} 010 \\ f_4 := 000, u_{\{0,1\}} \\ 001 \end{array} \right\} \xrightarrow{s} \begin{array}{l} 010 \\ 000 \\ 000 \end{array} =: g \quad g \text{ has core } \begin{array}{l} 01 \\ 00 \end{array} ([0, 1, 2] \mapsto [0, 1, 0])$$

$$\text{Implementation: } g(x, y) + 3 = \max_{z,w} [f_4(z, w) + f_4(w, y) + f_4(x, z) + u_{\{0,1\}}(z) + u_{\{0,1\}}(w)]$$

$$\left\{ \begin{array}{l} 011 \\ f_5 := 000, u_{\{0,1\}} \\ 001 \end{array} \right\} \xrightarrow{s} \begin{array}{l} 011 \\ 000 \\ 011 \end{array} =: g \quad g = f_2$$

$$\text{Implementation: } g(x, y) + 2 = \max_{z,w} [f_5(z, y) + f_5(w, z) + f_5(x, w) + u_{\{0,1\}}(z)]$$

$$\left\{ \begin{array}{l} 010 \\ f_6 := 100, u_{\{0,1\}} \\ 001 \end{array} \right\} \xrightarrow{s} \begin{array}{l} 010 \\ 100 \\ 000 \end{array} =: g \quad g \text{ has core } \begin{array}{l} 01 \\ 10 \end{array} ([0, 1, 2] \mapsto [0, 1, 0])$$

$$\text{Implementation: } g(x, y) + 3 = \max_{z,w} [f_6(z, w) + f_6(z, y) + f_6(w, x) + u_{\{0,1\}}(z)]$$

$$\left\{ \begin{array}{l} 011 \\ f_7 := 100 \\ 001 \end{array} \right\} \xrightarrow{s} \begin{array}{l} 010 \\ 101 \\ 101 \end{array} =: g \quad g = f_3$$

$$\text{Implementation: } g(x, y) + 2 = \max_{z,w} [f_7(w, z) + f_7(w, x) + f_7(y, z)]$$

$$\left\{ \begin{array}{l} 010 \\ f_8 := 001, u_{\{0,1\}} \\ 001 \end{array} \right\} \xrightarrow{s} \begin{array}{l} 000 \\ 100 \\ 100 \end{array} =: g \quad g \text{ has core } \begin{array}{l} 00 \\ 10 \end{array} ([0, 1, 2] \mapsto [0, 1, 1])$$

$$\text{Implementation: } g(x, y) + 3 = \max_{z,w} [f_8(z, w) + f_8(x, w) + f_8(y, z) + u_{\{0,1\}}(z)]$$

$$\left\{ \begin{array}{l} 010 \\ f_9 := 101 \\ 001 \end{array} \right\} \xrightarrow{s} \begin{array}{l} 010 \\ 101 \\ 101 \end{array} =: g \quad g = f_3$$

$$\text{Implementation: } g(x, y) + 2 = \max_{z,w} [f_9(w, z) + f_9(w, y) + f_9(x, z)]$$

$$\left\{ \begin{array}{l} 010 \\ f_{10} := 000, u_{\{0,1\}} \\ 101 \end{array} \right\} \xrightarrow{s} \begin{array}{l} 010 \\ 000 \\ 010 \end{array} =: g \quad g \text{ has core } \begin{array}{l} 01 \\ 00 \end{array} ([0, 1, 2] \mapsto [0, 1, 0])$$

$$\text{Implementation: } g(x, y) + 3 = \max_{z,w} [f_{10}(z, y) + f_{10}(w, z) + f_{10}(w, x) + u_{\{0,1\}}(z)]$$

$$\left\{ \begin{array}{l} 011 \\ f_{11} := 000, u_{\{0,1\}} \\ 101 \end{array} \right\} \xrightarrow{s} \begin{array}{l} 011 \\ 000 \\ 011 \end{array} =: g \quad g = f_2$$

$$\text{Implementation: } g(x, y) + 3 = \max_{z,w} [f_{11}(z, w) + f_{11}(z, y) + f_{11}(w, x) + u_{\{0,1\}}(z)]$$

$$\left\{ \begin{array}{l} 011 \\ f_{12} := 100, u_{\{0,1\}} \\ 101 \end{array} \right\} \xrightarrow{s} \begin{array}{l} 011 \\ 100 \\ 100 \end{array} =: g \quad g \text{ has core } \begin{array}{l} 01 \\ 10 \end{array} ([0, 1, 2] \mapsto [0, 1, 1])$$

Implementation:

$$g(x, y) + 4 = \max_{z,w} [f_{12}(z, w) + f_{12}(z, y) + f_{12}(w, x) + u_{\{0,1\}}(z) + u_{\{0,1\}}(w)]$$

$$\left\{ \begin{array}{l} f_{13} := \\ \begin{array}{l} 010 \\ 000 \\ 011 \end{array} \end{array}, u_{\{0,1\}} \right\} \xrightarrow{s} \begin{array}{l} 011 \\ 000 \\ 011 \end{array} =: g \quad g = f_2$$

Implementation: $g(x, y) + 2 = \max_{z,w} [f_{13}(z, w) + f_{13}(w, y) + f_{13}(x, z) + u_{\{0,1\}}(z)]$

$$\left\{ \begin{array}{l} f_{14} := \\ \begin{array}{l} 010 \\ 001 \\ 011 \end{array} \end{array}, u_{\{0,1\}} \right\} \xrightarrow{s} \begin{array}{l} 011 \\ 000 \\ 011 \end{array} =: g \quad g = f_2$$

Implementation: $g(x, y) + 3 = \max_{z,w} [f_{14}(z, w) + f_{14}(x, z) + f_{14}(y, w) + u_{\{0,1\}}(z)]$

$$\left\{ \begin{array}{l} f_{15} := \\ \begin{array}{l} 010 \\ 101 \\ 011 \end{array} \end{array}, u_{\{0,1\}} \right\} \xrightarrow{s} \begin{array}{l} 010 \\ 101 \\ 010 \end{array} =: g \quad g \text{ has core } \begin{array}{l} 01 \\ 10 \end{array} ([0, 1, 2] \mapsto [0, 1, 0])$$

Implementation:

$g(x, y) + 4 = \max_{z,w} [f_{15}(z, w) + f_{15}(z, y) + f_{15}(w, x) + u_{\{0,1\}}(z) + u_{\{0,1\}}(w)]$

Acknowledgments. The authors wish to thank Peter Jeavons and Benoit Larose for providing detailed comments on this paper.

REFERENCES

- [1] G. AUSIELLO, P. CRESZENZI, G. GAMBOSI, V. KANN, A. MARCHETTI-SPACCAMELA, AND M. PROTASI, *Complexity and Approximation*, Springer-Verlag, New York, 1999.
- [2] A. BULATOV, *A dichotomy theorem for constraints on a 3-element set*, in Proceedings of the 43rd Annual IEEE Symposium on Foundations of Computer Science (FOCS'02), IEEE Computer Society, Los Alamitos, CA, 2002, pp. 649–658.
- [3] A. BULATOV, *Mal'tsev constraints are tractable*, Technical report RR-02-05, Computing Laboratory, Oxford University, Oxford, UK, 2002.
- [4] A. BULATOV, *Tractable conservative constraint satisfaction problems*, in Proceedings of the 18th Annual IEEE Symposium on Logic in Computer Science (LICS'03), IEEE Computer Society, Los Alamitos, CA, 2003, pp. 321–330.
- [5] A. BULATOV AND V. DALMAU, *Towards a dichotomy theorem for the counting constraint satisfaction problem*, in Proceedings of the 44th Annual IEEE Symposium on Foundations of Computer Science (FOCS'03), IEEE Computer Society, Los Alamitos, CA, 2003, pp. 562–571.
- [6] R. E. BURKARD, B. KLINZ, AND R. RUDOLF, *Perspectives of Monge properties in optimization*, *Discrete Appl. Math.*, 70 (1996), pp. 95–161.
- [7] D. COHEN, M. COOPER, P. JEAVONS, AND A. KROKHIN, *Supermodular functions and the complexity of Max CSP*, *Discrete Appl. Math.*, 149 (2005), pp. 53–72.
- [8] D. COHEN, P. JEAVONS, AND M. GYSENS, *Closure properties of constraints*, *J. ACM*, 44 (1997), pp. 527–548.
- [9] N. CREIGNOU, *A dichotomy theorem for maximum generalized satisfiability problems*, *J. Comput. System Sci.*, 51 (1995), pp. 511–522.
- [10] N. CREIGNOU, S. KHANNA, AND M. SUDAN, *Complexity Classifications of Boolean Constraint Satisfaction Problems*, SIAM Monogr. Discrete Math. Appl. 7, SIAM, Philadelphia, 2001.
- [11] M. DATAR, T. FEDER, A. GIONIS, R. MOTWANI, AND R. PANIGRAHY, *A combinatorial algorithm for MAX CSP*, *Inform. Process. Lett.*, 85 (2003), pp. 307–315.
- [12] L. ENGBRETSSEN, *The nonapproximability of non-Boolean predicates*, *SIAM J. Discrete Math.*, 18 (2004), pp. 114–129.
- [13] L. ENGBRETSSEN AND V. GURUSWAMI, *Is constraint satisfaction over two variables always easy?*, *Random Structures Algorithms*, 25 (2004), pp. 150–178.
- [14] T. FEDER AND M. Y. VARDI, *The computational structure of monotone monadic SNP and constraint satisfaction: A study through Datalog and group theory*, *SIAM J. Comput.*, 28 (1998), pp. 57–104.
- [15] S. FUJISHIGE, *Submodular Functions and Optimization*, *Annals of Discrete Mathematics* 47, North-Holland, Amsterdam, 1991.
- [16] J. HÅSTAD, *Some optimal inapproximability results*, *J. ACM*, 48 (2001), pp. 798–859.
- [17] J. HÅSTAD, *Every 2-CSP allows nontrivial approximation*, in Proceedings of the 37th ACM Symposium on Theory of Computing (STOC'05), ACM, New York, 2005, pp. 740–746.
- [18] P. JONSSON, *Boolean constraint satisfaction: Complexity results for optimization problems with arbitrary weights*, *Theoret. Comput. Sci.*, 244 (2000), pp. 189–203.

- [19] S. KHANNA, M. SUDAN, L. TREVISAN, AND D. P. WILLIAMSON, *The approximability of constraint satisfaction problems*, SIAM J. Comput., 30 (2001), pp. 1863–1920.
- [20] C. LUND AND M. YANNAKAKIS, *The approximation of maximum subgraph problems*, in Automata, Languages, and Programming (ICALP'93), Lecture Notes in Comput. Sci. 700, Springer-Verlag, Berlin, 1993, pp. 40–51.
- [21] C. H. PAPADIMITRIOU AND M. YANNAKAKIS, *Optimization, approximation, and complexity classes*, J. Comput. System Sci., 43 (1991), pp. 425–440.
- [22] T. J. SCHAEFER, *The complexity of satisfiability problems*, in Proceedings of the 10th Annual ACM Symposium on Theory of Computing (STOC'78), ACM, New York, 1978, pp. 216–226.
- [23] M. SERNA, L. TREVISAN, AND F. XHAFI, *The (parallel) approximability of non-boolean satisfiability problems and restricted integer programming*, in Proceedings STACS'98, Lecture Notes in Comput. Sci. 1373, Springer-Verlag, Berlin, 1998, pp. 488–498.
- [24] D. TOPKIS, *Supermodularity and Complementarity*, Princeton University Press, Princeton, NJ, 1998.
- [25] L. TREVISAN, *Inapproximability of Combinatorial Optimization Problems*, Technical report 65, Electronic Colloquium on Computational Complexity, 2004.
- [26] L. TREVISAN, G. B. SORKIN, M. SUDAN, AND D. P. WILLIAMSON, *Gadgets, approximation, and linear programming*, SIAM J. Comput., 29 (2000), pp. 2074–2097.
- [27] D. ZUCKERMAN, *On unapproximable versions of NP-complete problems*, SIAM J. Comput., 25 (1996), pp. 1293–1304.
- [28] U. ZWICK, *Approximation algorithms for constraint satisfaction problems involving at most three variables per constraint*, in Proceedings of the 9th Annual ACM–SIAM Symposium on Discrete Algorithms (SODA'98), ACM, New York, SIAM, Philadelphia, 1998, pp. 201–210.

BALANCED ALLOCATIONS: THE HEAVILY LOADED CASE*

PETRA BERENBRINK[†], ARTUR CZUMAJ[‡], ANGELIKA STEGER[§], AND
BERTHOLD VÖCKING[¶]

Abstract. We investigate balls-into-bins processes allocating m balls into n bins based on the multiple-choice paradigm. In the classical *single-choice* variant each ball is placed into a bin selected uniformly at random. In a *multiple-choice* process each ball can be placed into one out of $d \geq 2$ randomly selected bins. It is known that in many scenarios having more than one choice for each ball can improve the load balance significantly. Formal analyses of this phenomenon prior to this work considered mostly the lightly loaded case, that is, when $m \approx n$. In this paper we present the first *tight* analysis in the *heavily loaded* case, that is, when $m \gg n$ rather than $m \approx n$.

The best previously known results for the multiple-choice processes in the heavily loaded case were obtained using majorization by the single-choice process. This yields an upper bound of the maximum load of bins of $m/n + \mathcal{O}(\sqrt{m \ln n / n})$ with high probability. We show, however, that the multiple-choice processes are fundamentally different from the single-choice variant in that they have “*short memory*.” The great consequence of this property is that the deviation of the multiple-choice processes from the optimal allocation (that is, the allocation in which each bin has either $\lfloor m/n \rfloor$ or $\lceil m/n \rceil$ balls) does not increase with the number of balls as in the case of the single-choice process. In particular, we investigate the allocation obtained by two different multiple-choice allocation schemes, the greedy scheme due to Azar et al. and the always-go-left scheme due to Vöcking. We show that these schemes result in a maximum load of only $m/n + \mathcal{O}(\ln \ln n)$ with high probability. All our detailed bounds on the maximum load are tight up to an additive constant.

Furthermore, we investigate the two multiple-choice algorithms in a comparative study. We present a majorization result showing that the always-go-left scheme obtains a better load balancing than the greedy scheme for any choice of n , m , and d .

Key words. occupancy problems, balls-into-bins processes, randomized resource allocation

AMS subject classifications. 60K30, 68W15, 68W20

DOI. 10.1137/S009753970444435X

1. Introduction. One of the central topics in the area of randomized algorithms is the study of *occupancy problems* for *balls-into-bins processes*; see, e.g., [1, 2, 3, 5, 6, 8, 9, 10, 11, 12, 15, 17, 20, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 33]. We consider allocation processes in which a set of independent balls representing, e.g., tasks or jobs is assigned to a set of bins representing, e.g., servers or machines. Since the framework of balls-into-bins processes can be used to translate realistic problems into a formal mathematical model in a natural way, it has been frequently analyzed in probability theory [15, 17], random graph theory, and, most recently, in computer science. For example, in theoretical computer science, the balls-into-bins processes found many applications in hashing (see, e.g., [16]) or randomized rounding. They also

*Received by the editors June 9, 2004; accepted for publication (in revised form) August 15, 2005; published electronically March 24, 2006. This research was supported in part by SBR grant 421090, NSF grant CCR-0105701, an NSERC-Discovery grant, DFG grant SFB 342, and DFG grant Vo889/1. An extended abstract of this paper appeared in *Proceedings of the 32nd Annual ACM Symposium on Theory of Computing*, Portland, OR, 2000, ACM Press, New York, 2000, pp. 745–754.

<http://www.siam.org/journals/sicomp/35-6/44435.html>

[†]School of Computing Science, Simon Fraser University, Burnaby, BC, V5A 1S6, Canada (petra@cs.sfu.ca).

[‡]Department of Computer Science, New Jersey Institute of Technology, Newark, NJ 07102 (czumaj@cis.njit.edu).

[§]Department of Computer Science, ETH Zürich, 8092 Zürich, Switzerland (steger@inf.ethz.ch).

[¶]Department of Computer Science, RWTH Aachen, 52056 Aachen, Germany (voecking@cs.rwth-aachen.de).

play a crucial role in load balancing and resource allocation in parallel and distributed systems (see, e.g., [1, 10, 18, 31, 32]). The goal of our study is to derive improved upper and lower bounds on the maximum load or even the entire distribution of the load in all bins for balls-into-bins processes in which each ball is placed in a load-adaptive fashion into one out of a small number of randomly chosen bins.

In the classical *single-choice process* (see, e.g., [15, 17, 30]), each ball is placed into a bin chosen *independently and uniformly at random (i.u.r.)*. For the case of n bins and $m \geq n \ln n$ balls it is well known that there exists a bin receiving $m/n + \Theta(\sqrt{m \ln n / n})$ balls (see, e.g., [30]). This result holds not only in expectation but even with high probability (w.h.p.).¹ Let the *maximum load* denote the number of balls in the fullest bin and let the *max height above average* denote the difference between the maximum load and the average number of balls per bin (which is m/n in our notation). Then the max height above average of the single choice algorithm is $\Theta(\sqrt{m \ln n / n})$, w.h.p. Observe that the deviation between the randomized single-choice allocation and the optimal allocation increases with the number of balls.

In this paper we investigate randomized multiple-choice allocation schemes (see, e.g., [1, 3, 11, 22, 23, 33]). The idea of *multiple-choice algorithms* is to reduce the maximum load by choosing a small subset of the bins for each ball at random and placing the ball into one of these bins. Typically, the ball is placed into a bin with a minimum number of balls among the d alternatives. It is well known that having more than one choice for each ball can improve the load balancing significantly [1, 33]. Previous analyzes, however, are only able to deal with the *lightly loaded case*, i.e., $m = \mathcal{O}(n)$; the bounds for $m \gg n$ are far off. Our *main contribution* is the first tight analysis for the *heavily loaded case*, i.e., when $m = \omega(n)$. We investigate two different kinds of well-known multiple-choice algorithms, the *greedy scheme* and the *always-go-left scheme*.

- Algorithm Greedy[d] was introduced and analyzed by Azar et al. in [1]. Greedy[d] chooses $d \geq 2$ locations for each ball i.u.r. from the set of bins. The m balls are inserted sequentially, one by one, and each ball is placed into the least loaded among its d locations (if several locations have the same minimum load, then the ball is placed into an arbitrary one among them). Azar et al. [1] show that the max height above average (and the maximum load) is $\ln \ln n / \ln d + \Theta(m/n)$, w.h.p.
- Algorithm Left[d] was introduced and analyzed by Vöcking in [33]. Let n be a multiple of $d \geq 2$. This algorithm partitions the set of bins into d groups of equal size. These groups are ordered from left to right. Left[d] chooses for each ball d locations: one location from each group is chosen i.u.r. The m balls are inserted one by one and each ball is placed into the least loaded among its d locations. If there are several locations having the same minimum load, the ball is always placed into the leftmost group containing one of these locations. Vöcking [33] proved, rather surprisingly, that the use of this unfair tie breaking mechanism leads to a better load balancing than a fair mechanism that resolves ties at random. In particular, the max height above average (and the maximum load) produced by Left[d] is only $\ln \ln n / (d \ln \phi_d) + \Theta(m/n)$, w.h.p., where $1.61 \leq \phi_d \leq 2$.

In the lightly loaded case, the bounds above are tight up to additive constants. In the heavily loaded case, however, these bounds are not even as good as the bounds

¹Throughout the entire paper we say an event A related to the process of allocating m balls into n bin occurs *with high probability (w.h.p.)* if $\Pr[A] \geq 1 - n^{-\kappa}$ for an arbitrarily chosen constant $\kappa \geq 0$. Notice that this probability does *not* depend on m , the number of balls.

known for the classical single-choice process. In fact, the best known bounds for the multiple-choice algorithms in the heavily loaded case are obtained using majorization from the single-choice process showing only that the multiple-choice algorithms do not behave worse than the single-choice process.

Unfortunately, the known methods for analyzing the multiple-choice algorithms do not allow us to obtain better results for the heavily loaded case. Both the techniques used in [1] (“layered induction”) and [33] (“witness trees”) inherently assume a load of $2m/n$ already in their base case and therefore they do not seem to be suitable to prove a bound better than $2m/n$. Alternative proof techniques using differential equations as suggested in [5, 22, 23, 34, 35] fail for the heavily loaded case, too. The reason is that the concentration results obtained by Kurtz’s theorem hold only for a limited number of balls. Therefore, the analysis of the heavily loaded case requires new ideas. Before we proceed with the detailed statement of our results we first provide some terminology.

1.1. Basic definitions and notation. We model the state of the system by *load vectors*. A load vector $u = (u_1, \dots, u_n)$ specifies that the number of balls in the i th bin (the *load* of the i th bin) is u_i . If u is *normalized*, then the entries in the vector are sorted in decreasing order so that u_i describes the number of balls in the i th fullest bin. In the case of Greedy[d], the order among the bins does not matter so that we can restrict the state space to normalized load vectors. In the case of Left[d], however, we need to consider general load vectors.

Suppose X_t denotes the load vector at time t , i.e., after inserting t balls using Greedy[d] or Left[d]. Then the stochastic process $(X_t)_{t \in \mathbb{N}}$ corresponds to a Markov chain $\mathfrak{M} = (\mathbf{X}_t)_{t \in \mathbb{N}}$ whose transition probabilities are defined by the respective allocation process. In particular, X_t is a random variable obeying some probability distribution $\mathcal{L}(X_t)$ defined by the allocation scheme. (Throughout the paper we use the standard notation to denote the probability distribution of a random variable U by $\mathcal{L}(U)$.) We use a standard measure of discrepancy between two probability distributions ϑ and ν on a space Ω , the *variation distance*, defined as

$$\|\vartheta - \nu\| = \frac{1}{2} \sum_{\omega \in \Omega} |\vartheta(\omega) - \nu(\omega)| = \max_{A \subseteq \Omega} (\vartheta(A) - \nu(A)).$$

A basic technique used in this paper is *coupling* (cf., e.g., [4, 7]). A coupling for two (possibly identical) Markov chains $\mathfrak{M}_X = (\mathbf{X}_t)_{t \in \mathbb{N}}$ with state space Ω_X and $\mathfrak{M}_Y = (\mathbf{Y}_t)_{t \in \mathbb{N}}$ with state space Ω_Y is a stochastic process $(X_t, Y_t)_{t \in \mathbb{N}}$ on $\Omega_X \times \Omega_Y$ such that each of $(X_t)_{t \in \mathbb{N}}$ and $(Y_t)_{t \in \mathbb{N}}$ is a faithful copy of \mathfrak{M}_X and \mathfrak{M}_Y , respectively.

Another basic concept that we use frequently is *majorization* (cf., e.g., [1]). We say that a vector $u = (u_1, \dots, u_n)$ is *majorized* by a vector $v = (v_1, \dots, v_n)$, written $u \leq v$, if for all $1 \leq i \leq n$ it holds that

$$\sum_{1 \leq j \leq i} u_{\pi(j)} \leq \sum_{1 \leq j \leq i} v_{\sigma(j)},$$

where π and σ are permutations of $1, \dots, n$ such that $u_{\pi(1)} \geq u_{\pi(2)} \geq \dots \geq u_{\pi(n)}$ and $v_{\sigma(1)} \geq v_{\sigma(2)} \geq \dots \geq v_{\sigma(n)}$. Given an allocation scheme \mathcal{X} defining a Markov chain $\mathfrak{M}_X = (\mathbf{X}_t)_{t \in \mathbb{N}}$ and an allocation scheme \mathcal{Y} defining a Markov chain $\mathfrak{M}_Y = (\mathbf{Y}_t)_{t \in \mathbb{N}}$, we say that \mathcal{X} is *majorized* by \mathcal{Y} if there is a coupling between the two Markov chains \mathfrak{M}_X and \mathfrak{M}_Y such that $X_t \leq Y_t$ for all $t \in \mathbb{N}$.

In order to express our results of the always-go-left scheme we use *d -ary Fibonacci numbers*. Define $F_d(k) = 0$ for $k \leq 0$, $F_d(1) = 1$, and $F_d(k) = \sum_{i=1}^d F_d(k-i)$ for $k \geq 2$.

Let $\phi_d = \lim_{k \rightarrow \infty} \sqrt[k]{F_d(k)}$. Then $F_d(k) = \Theta(\phi_d^k)$. Notice that ϕ_2 corresponds to the golden ratio. ϕ_d is called the *d-ary golden ratio*. In general $1.61 < \phi_2 < \phi_3 < \dots < 2$.

1.2. New results. Our main contribution is the first tight analysis for multiple-choice algorithms assuming an arbitrary number of balls. Our first result is a tight analysis of Greedy[d] in the heavily loaded case when the number of balls is upper-bounded by a polynomial in the number of bins.

LEMMA 1.1. *Let $\beta \geq 1$ be an arbitrary constant. Suppose we allocate m balls to n bins using Greedy[d] with $d \geq 2$ and $m \leq n^\beta$. Then the number of bins with load at least $\frac{m}{n} + i + \gamma$ is upper bounded by $n \cdot \exp(-d^i)$, w.h.p., where γ denotes a suitable constant, $\gamma = \gamma(\beta)$.*

Even if Lemma 1.1 may seem to be a simple extension of the analysis of the greedy algorithm with $m = \mathcal{O}(n)$ from [1], our analysis is significantly more complicated. The main idea behind the proof is similar to the layered induction approach from [1]. However, the fact that the number of balls is only bounded by a polynomial in the number of bins requires many additional tricks to be applied. In particular, unlike in [1], we have to consider the entire distribution of the bins in our analysis (while in [1] the bins with load smaller than the average could be ignored).

The techniques used to prove Lemma 1.1 cannot be extended to deal with the case when m is unbounded. This is because the analysis in the proof of Lemma 1.1 is based on an inductive argument showing that the bound given in the lemma holds after throwing *any* number $m' \leq m$ of the balls. Of course, this approach cannot work if m is unbounded, because in that case we expect that for some $m' \leq m$ there will be some bins having a huge load, even w.h.p. Therefore, to extend the result of Lemma 1.1 to all values of m we will need other techniques.

Our next result is the main technical contribution of this paper and is central for extending Lemma 1.1 to all values of m . It shows that the multiple-choice processes are fundamentally different from the classical single-choice process in that they have “short memory.”

LEMMA 1.2 (Short Memory Lemma). *Let $\varepsilon > 0$. Let $d \geq 2$ be any integer. Let X and Y be any two load vectors describing the allocation of m balls to n bins. Let X_t (Y_t) be the random variable that describes the load vector after allocating t further balls on top of X (Y , respectively) using protocol Greedy[d]. Then there is a $\tau = \mathcal{O}(m n^6 \ln^4(1/\varepsilon))$ such that $\|\mathcal{L}(X_\tau) - \mathcal{L}(Y_\tau)\| \leq \varepsilon$. In the special case when $d = 2$, this result holds even with $\tau = \mathcal{O}(m n^2 + n^4 \ln(m/\varepsilon))$.*

The proof of Lemma 1.2 is done by analyzing the mixing time of the underlying Markov chain describing the load distribution of the bins. Our study of the mixing time is via a new variant of the coupling method (called neighboring coupling (see Lemma 3.2 in section 3.1.2)) which may be of independent interest.

The Short Memory Lemma implies the following property of the Greedy[d] process (see Corollary 4.1 for a precise statement). Suppose that we begin in an arbitrary configuration with the load difference between any pair of bins being at most Δ . Then the Greedy[d] process “forgets” this unbalance in $\Delta \cdot \text{polylog}(\Delta) \cdot \text{poly}(n)$ steps. That is, the allocation after inserting further $\Delta \cdot \text{polylog}(\Delta) \cdot \text{poly}(n)$ balls is stochastically undistinguishable from an allocation obtained by starting from a totally balanced system. This is in contrast to the single-choice process that requires $\Delta^2 \cdot \text{poly}(n)$ steps to “forget” a load difference of Δ . We show that this property implies a fundamental difference between the allocation obtained by the multiple- and the single-choice algorithms. While the allocation of the single-choice algorithm deviates more and more from the optimal allocation with an increasing number of balls, the deviation between

the multiple-choice and the optimal allocation is independent of the number of balls. This allows us to concentrate the analysis for the large number of balls m on the case when m is only a polynomial of n .

Next, we show how to incorporate the results above, in Lemmas 1.1 and 1.2, to obtain our main result about the load distribution of Greedy[d].

THEOREM 1.3. *Let γ denote a suitable constant. Suppose we allocate m balls to n bins using Greedy[d] with $d \geq 2$. Then the number of bins with load at least $\frac{m}{n} + i + \gamma$ is upper bounded by $n \cdot \exp(-d^i)$, w.h.p.*

This result is tight up to additive constants in the sense that, for $m \geq n$, the number of bins with load at least $\frac{m}{n} + i \pm \Theta(1)$ is also bounded below by $n \cdot \exp(-d^i)$, w.h.p. In particular, Theorem 1.3 implies immediately the following corollary, which is tight up to a constant additive term.

COROLLARY 1.4. *If m balls are allocated into n bins using Greedy[d] with $d \geq 2$, then the number of balls in the fullest bin is $\frac{m}{n} + \frac{\ln \ln n}{\ln d} \pm \Theta(1)$, w.h.p. (that is, the max height above average is $\frac{\ln \ln n}{\ln d} \pm \Theta(1)$, w.h.p.).*

Next, we analyze the always-go-left scheme. The load distribution is described in terms of Fibonacci numbers defined in the previous section.

THEOREM 1.5. *Let γ denote a suitable constant. Suppose we allocate m balls into n bins using Left[d] with $d \geq 2$. Then the number of bins with load at least $\frac{m}{n} + i + \gamma$ is upper bounded by $n \cdot \exp(-\phi_d^{d^i})$, w.h.p.*

Also this bound is tight up to additive constants because the number of bins with load at least $\frac{m}{n} + i \pm \Theta(1)$ is lower-bounded by $n \cdot \exp(-\phi_d^{d^i})$, w.h.p., too. Similarly as in the case of the analysis of Greedy[d], Theorem 1.5 immediately implies a tight bound for the maximum load when using Left[d].

COROLLARY 1.6. *If m balls are allocated into n bins using Left[d] with $d \geq 2$, then the number of balls in the fullest bin is $\frac{m}{n} + \frac{\ln \ln n}{d \cdot \ln \phi_d} \pm \Theta(1)$, w.h.p. (that is, the max height above average is $\frac{\ln \ln n}{d \cdot \ln \phi_d} \pm \Theta(1)$, w.h.p.).*

In addition to these quantitative results, we investigate the relationship between the greedy and the always-go-left scheme directly.

THEOREM 1.7. *Left[d] is majorized by Greedy[d].*

In other words, we show that the always-go-left scheme produces a (stochastically) better load balancing than the greedy scheme for any possible choice of d , n , and m . We notice also that Theorem 1.7 is the key part of our analysis in Theorem 1.5.

1.3. Outline. In the first part of the paper we present the analysis of the greedy process. We begin in section 2 with the analysis of Greedy[d] for a polynomial number of balls (Lemma 1.1). Next, in section 3, we show that Greedy[d] has short memory (Lemma 1.2). Based on this property, we show in section 4 that our analysis for a polynomial number of balls can be extended to the analysis of the allocation for an arbitrary number of balls (Theorem 1.3).

In the second part of the paper we analyze the always-go-left process. Here we do not prove the short memory property explicitly. Instead, our main tool is majorization of Left[d] by Greedy[d]. In section 5, we show this majorization result, Theorem 1.7. In section 6, we analyze the allocation obtained by Left[d] based on the knowledge about the allocation of Greedy[d] to prove Theorem 1.5.

2. The behavior of Greedy[d] in the polynomially loaded case. In this section, we investigate the allocation obtained by Greedy[d] in the polynomially loaded case. In particular, we prove Lemma 1.1. In this theorem it is assumed that the number of balls is polynomially bounded by the number of bins, that is, $m \leq n^\delta$ with

$\delta > 0$ denoting an arbitrary constant. The theorem states that there exists a constant $\gamma \geq 0$ such that the number of bins with load at least $\frac{m}{n} + i + \gamma$ is at most $n \cdot \exp(-d^i)$, w.h.p. Recall that the term *w.h.p.* means that an event holds with probability at least $1 - n^{-\kappa}$ for any fixed $\kappa \geq 0$. Of course, the choice of γ has to depend on κ and δ . Observe that if $n < n_0$ for some constant n_0 , then the theorem is trivially satisfied by setting $\gamma = n_0^\delta$. We will use this observation at several points in our analysis at which certain inequalities hold only under the assumption $n \geq n_0$ for a suitable choice of n_0 . The choice of n_0 might depend only on δ and κ .

Without loss of generality, we assume that $m \leq n^\delta$ is a multiple of n . We prove the theorem by induction. For this purpose, we divide the set of balls into at most $n^{\delta-1}$ *batches* of size n each. The allocation *at time* t describes the number of balls in the bins after we have inserted the balls of the first t batches, i.e., after placing tn balls, starting with a set of empty bins at time 0. Our proof is by induction on $t \geq 0$. We provide several invariants characterizing a typical distribution of the balls among the bins at time t . We prove by induction that if these invariants hold before allocating a new batch, then they hold also after allocating the batch with probability at least $1 - n^{-\kappa}$ for any given $\kappa > 0$. This implies that the invariants hold with probability at least $1 - n^{-\kappa+(\delta-1)}$ after inserting the last batch because the number of batches is upper-bounded by $n^{\delta-1}$. In other words, we prove that each individual induction step holds w.h.p. which implies that the invariants hold w.h.p. over all steps because the number of induction steps is polynomially bounded.

2.1. Invariants for Greedy[d]. The average number of balls per bin at *time* t (that is, after allocating tn balls) is t . The bins with less than t balls are called *underloaded bins* and the bins with more than t balls are called *overloaded bins*. The *number of holes* at time t is defined as the minimal number of balls one would need to add to the underloaded bins so that each bin has load at least t . The *height* of a ball in a bin is defined such that the i th ball allocated to a bin has height i . We investigate the number of holes in the underloaded bins and the number of balls in the overloaded bins. In particular, we show that the following invariants hold w.h.p.

- $L(t)$: At time t , there are at most $0.74n$ holes.
- $H(t)$: At time t , there are at most $0.27n$ balls of height at least $t + 3$.

We prove these invariants by an interleaved induction; that is, the proof for $L(t)$ assumes $H(t-1)$ and the proof for $H(t)$ assumes $L(t)$. Notice that since t is the average number of balls per bin at time t , the number of holes at time t corresponds to the number of balls above height t . Thus, invariant $L(t)$ implies that there are at most $0.74n$ balls with height $t + 1$ or larger at time t . This property will enable us to show the upper bound on the number of balls with large height given in $H(t)$. In turn, we will see that there is a way to translate the upper bound on the number of balls with large height given in $H(t-1)$ into an upper bound on the number of holes, which will enable us to prove invariant $L(t)$.

Observe that the two invariants above do not directly imply Lemma 1.1. However, they will allow us to split the analysis into two parts: one for the underloaded bins and one for the overloaded bins. These two parts depend on each other only through the invariants L and H . In both of these parts we will specify further invariants. Finally, the invariants for the overloaded bins will imply Theorem 1.1.

Throughout the analysis, we use the following notation. For $i, t \geq 0$, $\alpha_i^{(t)}$ denotes the fraction of bins with load at most $t - i$ at time t , and $\beta_i^{(t)}$ denotes the fraction of bins with load at least $t + i$ at the same time. Figure 1 illustrates this notation.

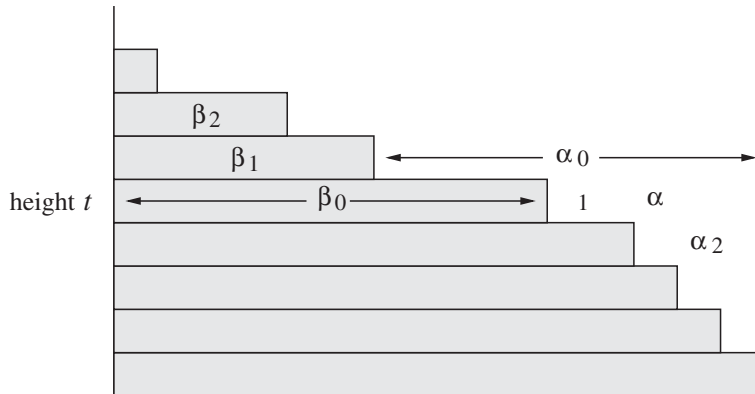


FIG. 1. Illustration of the terms $\alpha_i^{(t)}$ and $\beta_i^{(t)}$.

2.2. Analysis of the underloaded bins. In this section, we analyze the number of holes in the underloaded bins. We prove the following two invariants for time $t \geq 0$. Let c_1 and c_2 denote suitable constants with $c_1 \leq c_2$.

- $L_1(t)$: For $1 \leq i \leq c_1 \ln n$, $\alpha_i^{(t)} \leq 1.3 \cdot 2.8^{-i}$.
- $L_2(t)$: For $i \geq c_2 \ln n$, $\alpha_i^{(t)} = 0$.

Observe that the invariants $L_1(t)$ and $L_2(t)$ imply the invariant $L(t)$ as the number of holes at time t is at most

$$\sum_{i=1}^{\lfloor c_2 \ln n \rfloor} 1.3 \cdot 2.8^{-\min(i, \lceil c_1 \ln n \rceil)} \cdot n \leq 0.74n,$$

where the last inequality holds if $n \geq n_0$ for suitably chosen constant term n_0 . In the following, we prove that $L_1(t)$ and $L_2(t)$ hold w.h.p. Our analysis is focused on Greedy[2]; that is, we explicitly prove that the invariants $L_1(t)$ and $L_2(t)$ hold for Greedy[2]. Given that the invariants hold for $d = 2$, a majorization argument [1, Theorem 3.5] implies that the same invariants hold for every $d \geq 2$. In fact, the same argument shows that the choice of the tie breaking mechanism of Greedy[2] is irrelevant. Therefore, without loss of generality, we can assume that Greedy[2] breaks ties among bins of the same height by flipping a fair coin. Under this assumption, we have the following lemma.

LEMMA 2.1. *Let ℓ be an arbitrary integer and assume that at some point in time there exist (at most) $a_\ell n$ bins with at most ℓ balls and (at most) $a_{\ell-1} n$ bins with at most $\ell - 1$ balls. Suppose that b is a bin with load exactly ℓ . Then, the probability that the next ball allocated by Greedy[2] will be placed into bin b is (at least) $(2 - a_\ell - a_{\ell-1})/n$.*

Proof. Since the term $(2 - a_\ell - a_{\ell-1})/n$ is decreasing in both a_ℓ and $a_{\ell-1}$, we can assume without loss of generality that there are exactly $a_\ell n$ bins with at most ℓ balls and exactly $a_{\ell-1} n$ bins with at most $\ell - 1$ balls. The probability that the ball goes to one of the bins with load ℓ is

$$(a_\ell - a_{\ell-1})^2 + 2(a_\ell - a_{\ell-1})(1 - a_\ell) = (a_\ell - a_{\ell-1}) \cdot (2 - a_\ell - a_{\ell-1}),$$

because this event happens if and only if either both random locations of the ball point to a bin with load ℓ or at least one of them points to a bin with load ℓ and the

other to a bin with load larger than ℓ . Now, since we assume a random tie breaking mechanism, each of the bins with load ℓ is equally likely to receive the ball. Thus, given that the ball falls into one of these bins, the probability that b receives the ball is $\frac{1}{(a_\ell - a_{\ell-1})n}$ because the number of bins with load ℓ is $(a_\ell - a_{\ell-1})n$. Multiplying the two probabilities yields the lemma. \square

Combining Lemma 2.1 with invariant L_1 , we can now analyze the probability that a ball from batch t falls into a fixed bin b with a given number of holes. Applying invariant $L_1(t - 1)$, there are at most $\alpha_i^{(t-1)}n \leq 1.3 \cdot 2.8^{-i} \cdot n$ bins with load at most $(t - 1) - i$ at time $t - 1$ for every $1 \leq i \leq c_1 \ln n$. This upper bound not only holds at the time immediately before the first ball of batch t is inserted but can be applied to any ball from this batch since the load of a bin is nondecreasing over time. Now, applying Lemma 2.1 yields that the probability that a ball from batch t is assigned to a bin with load at most $(t - 1) - i$ is at least

$$\frac{2 - \alpha_i^{(t-1)} - \alpha_{i+1}^{(t-1)}}{n} \geq \frac{2 - 1.3 \cdot 2.8^{-i} - 1.3 \cdot 2.8^{-(i+1)}}{n}.$$

For $i \geq 3$, this probability is larger than $1.9/n$, which yields the following observation.

OBSERVATION 2.2. *The probability that a ball from batch t goes to any fixed bin with load at most $t - 4$ at the ball's insertion time is at least $1.9/n$, unless invariant $L_1(t - 1)$ fails.*

Thus bins with load $t - 4$ or less have almost twice the probability of receiving a ball than the average. This might give an intuition as to why none of the bins falls far behind. The following analysis puts this intuition into formal arguments.

LEMMA 2.3. *Let $t \geq 0$. Suppose the probability that one of the invariants $L_1(0), \dots, L_1(t - 1)$ fails is at most $n^{-\kappa'}$ for $\kappa' \geq 1$. For any fixed $\kappa > 0$, there exist constants c_0, c_1, c_2, c_3 (solely depending on κ) such that*

- *there are at most $n \cdot 0.18 \cdot 3^{-i+2}$ bins containing at most $t - i$ balls, for $c_0 < i \leq c_1 \ln n$, and*
- *every bin contains at least $t - c_2 \ln n$ balls,*

with probability at least $1 - n^{-\kappa} - n^{-\kappa'}$, provided $n \geq c_3$.

Proof. Consider a bin b . For any integer $s \geq 0$, let ℓ_s denote the load of bin b at time s , and let q_s be the number of holes in bin b at time s ; that is, $q_s = s - \ell_s$. Now, suppose $q_t \geq i + 4$, for $i \geq 0$. Since the number of holes can increase by at most one during the allocation of a batch, there exists an integer t' , $0 \leq t' < t$, such that $q_{t'} = 4$ and for all $s \in \{t' + 1, \dots, t\}$ it holds that $q_s \geq 4$. Observe that by this definition of t' , the number of balls from the batches $t' + 1, \dots, t$ that are assigned to bin b is at most $t - t' - i$.

The definition of t' implies that the bin b has at least four holes at any time during the allocation of the batches $t' + 1, \dots, t$. More formally, at the insertion time of any ball from batch $s \in \{t' + 1, \dots, t\}$, the bin b contains at most $s - 4$ balls. Thus, by Observation 2.2, it follows that every ball from these batches has probability at least $1.9/n$ to be assigned to bin b or there exist $s < t$ such that invariant $L_1(s)$ fails. Ignoring the latter event, the number of balls from these batches that are placed into bin b is stochastically dominated by a binomially distributed random variable $B((t - t')n, 1.9/n)$. However, we cannot simply condition on $L_1(0, \dots, t - 1)$ as this gives an unwanted bias to the random experiments under consideration. Instead we explicitly exclude the event $\neg L_1(0, \dots, t - 1)$ from our considerations. This way, we

obtain

$$\begin{aligned} \Pr[(q_t \geq i + 4) \wedge L_1(0, \dots, t - 1)] &\leq \sum_{t'=0}^{t-1} \Pr[\mathbf{B}((t - t')n, 1.9/n) \leq t - t' - i] \\ &\leq \sum_{\tau=1}^{\infty} \Pr[\mathbf{B}(\tau n, 1.9/n) \leq \tau - i] \\ &= \sum_{\tau=1}^{\infty} \sum_{k=i}^{\tau} \Pr[\mathbf{B}(\tau n, 1.9/n) = \tau - k]. \end{aligned}$$

Next, for $0 \leq k < \tau$, we obtain

$$\begin{aligned} \Pr[\mathbf{B}(\tau n, 1.9/n) = \tau - k] &= \binom{\tau n}{\tau - k} \cdot \left(1 - \frac{1.9}{n}\right)^{\tau n - \tau + k} \cdot \left(\frac{1.9}{n}\right)^{\tau - k} \\ &\leq \frac{(1.9 e \tau)^{\tau - k}}{(\tau - k)!} e^{-(\tau n - \tau + k) 1.9/n} \\ &\leq \frac{(1.9 \tau)^{\tau - k}}{(\tau - k)^{\tau - k}} e^{-0.9 \tau - k + 1.9 \tau/n} \\ &\leq \left(\frac{1.9 \tau}{\tau - k}\right)^{\tau - k} e^{-0.89 \tau - k}, \end{aligned}$$

where the last inequality holds for $n \geq 190$. Now, set $z = \tau/k > 1$. Then, we obtain for $k > 0$

$$\begin{aligned} \Pr[\mathbf{B}(\tau n, 1.9/n) = \tau - k] &\leq \left(\frac{1.9 z k}{(z - 1) k}\right)^{(z-1)k} e^{-0.79 z k - 0.1 \tau - k} \\ &= \left(\frac{1.9 z}{z - 1}\right)^{(z-1)k} e^{-0.79 (z-1)k - 1.79 k - 0.1 \tau} \\ &= \left(e^{-1.79} \left(\frac{1.9 z}{e^{0.79} (z - 1)}\right)^{(z-1)k}\right)^k e^{-0.1 \tau}. \end{aligned}$$

The term $\left(\frac{1.9 z}{e^{0.79} (z - 1)}\right)^{(z-1)}$, $z \geq 1$, takes its maximum at $z = 2.22\dots$ and is bounded from above by 1.74. Therefore,

$$\Pr[\mathbf{B}(\tau n, 1.9/n) = \tau - k] \leq (1.74 \cdot e^{-1.79})^k \cdot e^{-0.1 \tau} \leq 3.4^{-k} \cdot e^{-0.1 \tau}$$

for $0 < k < \tau$. The same inequality also holds for $k = 0$, because for $n \geq 13$ it holds that

$$\begin{aligned} \Pr[\mathbf{B}(\tau n, 1.9/n) = \tau - 0] &= \Pr[\mathbf{B}(\tau n, 1.9/n) = \tau] \\ &= \binom{\tau n}{\tau} \cdot \left(1 - \frac{1.9}{n}\right)^{\tau(n-1)} \cdot \left(\frac{1.9}{n}\right)^{\tau} \\ &\leq \left(\frac{e \tau n}{\tau}\right)^{\tau} e^{1.9 \tau \frac{n-1}{n}} \cdot \left(\frac{1.9}{n}\right)^{\tau} \\ &= \left(\frac{1.9 e}{e^{1.9 \frac{n-1}{n}}}\right)^{\tau} \\ &\leq e^{-0.1 \tau}. \end{aligned}$$

Finally, we can obtain the same bound for $k = \tau$. In this case,

$$\begin{aligned} \Pr[\mathbf{B}(\tau n, 1.9/n) = \tau - k] &= \Pr[\mathbf{B}(\tau n, 1.9/n) = 0] \\ &= \left(1 - \frac{1.9}{n}\right)^{\tau n} \\ &\leq e^{-1.9\tau} \\ &\leq 3.4^{-k} \cdot e^{-0.1\tau}. \end{aligned}$$

Substituting this bound into the upper bound for $\Pr[q_t \geq i + 4]$ gives

$$\Pr[(q_t \geq i + 4) \wedge L_1(0, \dots, t - 1)] \leq \sum_{\tau \geq 1} \sum_{k \geq i} 3.4^{-k} \cdot e^{-0.1\tau} \leq 13.5 \cdot 3.4^{-i}.$$

Now let Q_t denote the maximum number of holes over all bins at time t . Recall that the probability for the event $\neg L_1(0, \dots, t - 1)$ is bounded from above by $n^{-\kappa'}$. Consequently,

$$\Pr[Q_t \geq i + 4] \leq n^{-\kappa'} + \Pr[(Q_t \geq i + 4) \wedge L_1(0, \dots, t - 1)] \leq n^{-\kappa'} + n \cdot 13.5 \cdot 3.4^{-i}.$$

It follows $Q_t = \mathcal{O}(\log n)$, w.h.p. More specifically, for every $\kappa \geq 0$, there exists a constant c_2 such that every bin contains at least $t - c_2 \ln n$ balls, with probability $1 - n^{-\kappa'} - \frac{1}{2}n^{-\kappa}$. This yields the second statement given in the lemma. In the following, we show that the first statement holds with probability at least $1 - \frac{1}{2}n^{-\kappa}$. In particular, we prove that for any given $\kappa > 0$, there are constants c_0, c_1, c_3 such that, for $c_0 < i \leq c_1 \ln n$ and $n \geq c_3$, there are at most $0.18 \cdot 3^{-i+2} \cdot n$ bins containing $t - i$ or less balls at time t , with probability at least $1 - n^{-\kappa-1} \geq 1 - (2c_1 n^\kappa \ln n)^{-1}$. This way, the probability that one of the statements listed in the lemma fails is at most $n^{-\kappa'} + n^{-\kappa}$, so that the lemma is shown.

Let $c_0 = 40$. For $i > c_0$, we obtain

$$\Pr[q_t \geq i] \leq n^{-\kappa'} + 13.5 \cdot 3.4^{-i+4} \leq n^{-1} + 0.65 \cdot 2.8^{-i}.$$

Now let X_b be an indicator random variable that is 1 if bin b holds at most $t - i$ balls, and that is 0, otherwise. Define $X = \sum X_b$. We have to prove that $X \leq 1.3 \cdot 2.8^{-i} \cdot n$, w.h.p.

Let us first notice that

$$\mathbf{E}[X] \leq n \cdot \Pr[q_t \geq i] \leq 0.65 \cdot 2.8^{-i} \cdot n + 1.$$

The random variables X_b are “negatively associated” in the sense of [14, Proposition 7]. Hence, we can apply a Chernoff bound to these variables: For every $\mu \geq \mathbf{E}[X]$, $\Pr[X \geq 2\mu] \leq e^{-\mu/2}$. We choose $\mu = 0.65 \cdot 2.8^{-i} \cdot n + 1$, set $c_1 = 0.5$, and assume that c_3 is sufficiently large so that $\mu \geq 2(\kappa + 1) \ln n$ for every $i \leq c_1 \ln n$ and $n \geq c_3$. This yields

$$\Pr[X \geq 1.3 \cdot 2.8^{-i} \cdot n + 2] \leq \Pr[X \geq 2\mu] \leq e^{-\mu/2} \leq e^{-(\kappa+1) \ln n} = n^{-\kappa-1}.$$

This completes the proof of Lemma 2.3. \square

The second part of the lemma corresponds to invariant $L_2(t)$, and the first part corresponds to invariant $L_1(t)$, but only for $i > c_0$. Thus, it remains to show invariant $L_1(t)$ for $1 \leq i \leq c_0$; that is, we have to show $\alpha_i^t \leq 1.3 \cdot 2.8^{-i}$. We will solve this

problem by specifying a recursive formula upper-bounding the term $\alpha_i^{(t)}$, $1 \leq i \leq c_0$, in terms of the vector $(\alpha_{i-1}^{(t-1)}, \dots, \alpha_{i+3}^{(t-1)})$.

LEMMA 2.4. *Let $\epsilon > 0$ and a_0, \dots, a_4 be constant reals with $0 < a_4 < \dots < a_0 < 1$. Let k be a constant integer. Let ℓ denote any integer. Suppose for $i = 0, \dots, 4$ there are at most $a_i n$ bins with load at most $\ell - i$ at time $t - 1$. Then, at time t , the number of bins with load at most ℓ is at most $g_k(0) \cdot n$, w.h.p., where the function g is defined by*

$$g_j(i) = \begin{cases} a_i & \text{if } j = 0 \text{ or } i = 4, \\ (1 + \epsilon) \cdot (g_{j-1}(i + 1) + (g_{j-1}(i) - g_{j-1}(i + 1)) \cdot E) & \text{otherwise,} \end{cases}$$

where

$$E = \exp\left(-\frac{2 - g_{j-1}(i + 1) - g_{j-1}(i)}{k}\right).$$

Proof. For the time being, let us assume that n is a multiple of k . We divide the allocation of the n balls into k phases, into each of which we insert n/k balls using Greedy[2]. For $0 \leq i \leq 4$ and $1 \leq j \leq k$, we show that $n \cdot g_j(i)$ is an upper bound on the number of bins with load at most $\ell - i$ after phase j . Observe that this claim is trivially satisfied for $i = 4$.

We perform an induction on j , the index of the phase. Observe that for $0 \leq i \leq 4$, $n \cdot g_0(i)$ is an upper bound on the number of bins with load at most $\ell - i$ at the beginning of phase 1. In the inductive hypothesis we assume that $n \cdot g_{j-1}(i)$ is an upper bound on the number of bins with load at most $\ell - i$ at the beginning of phase $j \geq 1$. Now consider the allocation of the n/k balls in phase j . Suppose b is a bin having load $\ell - i$ ($0 \leq i \leq 3$) at the beginning of that phase. Lemma 2.1 implies that the probability that b receives none of the next n/k balls is at most

$$\left(1 - \frac{2 - g_{j-1}(i) - g_{j-1}(i + 1)}{n}\right)^{n/k} \leq \exp\left(-\frac{2 - g_{j-1}(i + 1) - g_{j-1}(i)}{k}\right) = E.$$

Thus, the expected number of bins with load exactly $\ell - i$ not receiving a ball in this phase is at most

$$n \cdot (g_{j-1}(i) - g_{j-1}(i + 1)) \cdot E.$$

As a consequence, the expected fraction of bins containing at most $\ell - i$ balls at the end of phase k is upper-bounded by

$$g_{j-1}(i + 1) + (g_{j-1}(i) - g_{j-1}(i + 1)) \cdot E$$

for $0 \leq i \leq 3$. This term, however, by the definition, is equivalent to $g_j(i)/(1 + \epsilon)$, and hence the expected number of bins containing at most $\ell - i$ balls is upper-bounded by $n g_j(i)/(1 + \epsilon)$. Applying Azuma's inequality to this expectation, we can observe that the deviation is at most $o(n)$, w.h.p. Hence, since $n \cdot g_j(i) \geq a_4 \cdot n = \Theta(n)$, we conclude that the stochastic deviation can be bounded by a factor $(1 + \epsilon)$ for any $\epsilon > 0$, provided n is sufficiently large. We conclude that the fraction of bins containing at most $\ell - i$ balls at the end of phase k is at most $g_j(i)$, w.h.p.

Now let us consider the case that n is not a multiple of k . In this case, we can upper-bound the probability that b receives none of the at least $n/k - 1$ balls from the next batch by

$$\left(1 - \frac{2}{n}\right)^{-1} \cdot E$$

instead of E as before. Obviously the leading factor in front of the E can be made arbitrarily small by choosing n sufficiently large so that it is covered by the $1 + \epsilon$ factor that we already used above to cover stochastic deviations. This completes the proof of the lemma. \square

In the following, we apply the recursive formula given in Lemma 2.4 to prove that invariant $L_1(t)$ holds for every $i \in \{1, \dots, c_0 - 1\}$. The recursion gives an upper bound for $\alpha_i^{(t)}$ in terms of the vector $(a_0, \dots, a_4) = (\alpha_{i-1}^{(t-1)}, \dots, \alpha_{i+3}^{(t-1)})$. For $i \geq 2$, the terms $\alpha_{i-1}^{(t-1)}, \dots, \alpha_{i+3}^{(t-1)}$ can be estimated using invariant $L_1(t-1)$. Suppose this invariant is given. Fix any $i \in \{2, \dots, c_0 - 1\}$. For $i' = 0, \dots, 4$, we set $a_{i'} = 1.3 \cdot 2.8^{-(i+i'-1)}$. Then, $a_{i'}$ is an upper bound on the fraction of bins with load at most $i - i'$ at time $t - 1$. Now we choose $k = 20$ and $\epsilon = \frac{1}{1000}$, and we numerically calculate $g_k(0)$ using Maple. For such a choice of parameters, we obtain $g_k(0) \leq 1.3 \cdot 2.8^{-i}$. By Lemma 2.4, $g_k(0)$ is an upper bound on $\alpha_i^{(t)}$, w.h.p. Thus, invariant $L(t)$ is shown for $i \geq 2$. Unfortunately, this approach does not work for $i = 1$, because in that case a_0 corresponds to $\alpha_0^{(t-1)}$, which is not covered by invariant $L_1(t-1)$. In what follows, we prove another lemma that gives an upper bound on $\alpha_0^{(t-1)}$ based on invariant $H(t-1)$.

LEMMA 2.5. *Suppose $H(t-1)$ is fulfilled. Let $(a_0, \dots, a_4) := (\alpha_0^{(t-1)}, \dots, \alpha_4^{(t-1)})$. Then*

$$a_0 \leq 1 - \frac{a_1 + a_2 + a_3 + a_4 - 0.27}{2}.$$

Proof. At any time $\tau \geq 0$, the number of holes at time τ is $A_\tau = \sum_{j \geq 1} \alpha_j^{(\tau)} n$. Since the number of balls above the average height is equal to the number of holes, we can conclude that A_τ also corresponds to the number of balls with height at least $\tau + 1$ at time τ . Now, suppose invariant $H(\tau)$ holds. Then, there are at most $B_\tau = 0.27n$ balls of height at least $\tau + 3$ at time τ . Combining these two bounds, the number of balls with height either $\tau + 1$ or $\tau + 2$ is lower-bounded by $A_\tau - B_\tau$. This implies that at least $(A_\tau - B_\tau)/2$ bins contain more than τ balls at time τ . As a consequence, the number of bins containing at most τ balls is upper-bounded by $n - (A_\tau - B_\tau)/2$. Hence,

$$\alpha_0^{(\tau)} n \leq n - \frac{A_\tau - B_\tau}{2} \leq n \cdot \left(1 - \frac{\sum_{j=1}^4 \alpha_j^{(\tau)} - 0.27}{2} \right).$$

Finally, setting $\tau = t - 1$ and $\alpha_j^{(\tau)} = a_j$ gives the lemma. \square

Now, we are ready to prove invariant $L_1(t)$ by showing $g_k(0) \leq 1.3 \cdot 2.8^{-1}$ for all choices of $a_{i'} \in [0, 1.3 \cdot 2.8^{-i'}]$, $1 \leq i' \leq 4$, and $a_0 \in [0, 1 - \frac{1}{2}(a_1 + a_2 + a_3 + a_4 - 0.27)]$. Again, we check the condition on $g_k(0)$ numerically using Maple. For this purpose we need to discretize the domains of the a_i 's. For the discretization, we use the monotonicity of $g_k(0)$: the term $g_k(0)$ is monotonically increasing in each of the terms a_0, \dots, a_4 . Therefore, it suffices to check the parameters a_1, \dots, a_4 in discrete steps of a suitable size $\delta > 0$ while assuming

$$a_0 = 1 - \frac{a_1 + a_2 + a_3 + a_4 - 0.27 - 4\delta}{2}.$$

In fact, we choose $k = 20$, $\epsilon = \frac{1}{1000}$, and $\delta = \frac{1}{400}$. The numerical calculations with Maple show that the above condition on $g_k(0)$ is satisfied in all cases. Hence, the invariants $L_1(t)$, $L_2(t)$, and, thus, $L(t)$ are shown.

2.3. Analysis of the overloaded bins. In this section, we will analyze the distribution of load in the overloaded bins. In particular, we prove invariant $H(t)$ stating that the number of balls with height at least $t + 3$ is at most $0.27n$. Our analysis is based on invariant $L(t)$ which we proved in the previous section based on $H(t - 1)$. Thus $H(t - 1)$ yields $L(t)$ and, in turn, $L(t)$ yields $H(t)$.

Our analysis of invariant $H(t)$ is obtained by the analysis of two further invariants that will imply invariant $H(t)$ and will yield the proof of Lemma 1.1. These invariants are defined as follows. Let

$$h(i) = 0.7 \cdot 0.53^{d^{i-2}}.$$

Let ℓ denote the smallest integer i such that $h(i) \leq n^{-0.9}$. Let $b \geq 1$ denote a suitable constant, whose value will be specified later. For $i \geq 3$, define

$$f(i) = \begin{cases} h(i) & \text{for } 2 \leq i < \ell, \\ \max\{h(i), \frac{1}{4}n^{-0.9}\} & \text{for } i = \ell, \\ bn^{-1} & \text{for } i = \ell + 1. \end{cases}$$

We will prove that the following invariants hold w.h.p.

- $H_1(t)$: $\beta_i^{(t)} \leq f(i)$ for $2 \leq i \leq \ell$,
- $H_2(t)$: $\sum_{i>\ell} \beta_i^{(t)} \leq bn^{-1}$.

Roughly speaking, invariant H_1 states that the sequence $\beta_2, \beta_3, \dots, \beta_\ell$ decreases doubly exponentially, and the number of balls on layer ℓ is upper-bounded by $\frac{1}{4}n^{0.1}$. Furthermore, invariant H_2 states that there is only a constant number of balls with a height larger than ℓ . These two invariants imply the bounds given in Lemma 1.1. Furthermore, these invariants imply the invariant $H(t)$ as they upper-bound the number of balls above layer $t + 3$ by

$$\sum_{i=3}^{\ell-1} h(i)n + \frac{1}{4}n^{-0.1} + b \leq 0.26n + \frac{1}{4}n^{-0.1} + b \leq 0.27n,$$

where the last inequality holds for $n \geq 50b + 78$. We show the invariants H_1 and H_2 by induction. Our induction assumptions are $H_1(0), \dots, H_1(t - 1)$, $H_2(t - 1)$, and $L(t)$. We prove that these assumptions imply $H_1(t)$, $H_2(t)$, and $H(t)$, w.h.p. Our analysis starts by summarizing some properties of the function f . We assume that n is sufficiently large.

CLAIM 2.6.

- A1. $f(2) = 0.371$;
- A2. $f(i) \geq 0.53^{-2}f(i + 1)$ for $3 \leq i \leq \ell$;
- A3. $f(i) \geq 0.7^{-1}f(i - 1)^d$ for $3 \leq i \leq \ell$;
- A4. $f(i) \geq \frac{1}{4}n^{-0.9}$ for $3 \leq i \leq \ell$.

Proof. The properties A1 and A4 follow directly from the definition of f . Property A2 can be seen as follows. For $3 \leq i \leq \ell - 2$, $f(i) = h(i)$ as well as $f(i + 1) = h(i + 1)$. Thus,

$$f(i + 1) = 0.7 \cdot 0.53^{d^{i-1}} = 0.7 \cdot 0.53^{d^{i-2} \cdot d} \leq 0.7 \cdot 0.53^{d^{i-2} + 2} \leq 0.53^2 \cdot f(i).$$

If $i = \ell - 1$, then $f(i + 1) = h(i + 1)$ or $f(i + 1) = \frac{1}{4}n^{-0.9}$. In the former case, we can apply the same argument as before. In the latter case, we apply $f(i) \geq n^{-0.9}$ which immediately yields $f(i)/f(i + 1) > 4 \geq 0.53^{-2}$. Finally, in the case $i = \ell$, we need the

assumption that n is sufficiently large so that $f(\ell)/f(\ell + 1) \geq \frac{1}{4}n^{-0.9}/(bn) \geq 0.53^{-2}$. It remains to prove property A3. For $3 \leq i \leq \ell$,

$$f(i - 1)^d = (0.7 \cdot 0.53^{d^{i-3}})^d = 0.7^d \cdot 0.53^{d^{i-2}} \leq 0.7 \cdot f(i).$$

Now we use these properties to show $H_1(t)$ using a “layered induction” on the index i , similar to the analysis presented in [1]. For the base case, i.e., $i = 2$, we apply invariant $L(t)$. This invariant yields that, at time t , there exist at most $0.74n$ balls of height larger than t . Consequently, the number of bins with $t + 2$ or more balls is at most $0.74n/2 = 0.37n$. Applying property A1 yields $\beta_2^{(t)} \leq 0.37 \leq f(2)$. Thus, invariant $H_1(t)$ is shown for $i = 2$.

Next we prove $H_1(t)$ for $i \in \{3, \dots, \ell\}$. Fix i , $3 \leq i \leq \ell$. We assume that $H_1(t)$ holds for $i - 1$. Let q denote the number of bins that contain $t + i$ or more balls at time $t - 1$, i.e., immediately before batch t is inserted, and let p denote the number of balls from batch t that are placed into a bin containing at least $t + i - 1$ balls at time t . Observe that $\beta_i^{(t)} \cdot n \leq q + p$. Thus, it suffices to show $q + p \leq f(i) \cdot n$. Applying induction assumption $H_1(t - 1)$, we obtain

$$q \leq \beta_{i+1}^{(t-1)} \cdot n \leq f(i + 1) \cdot n \stackrel{(A2)}{\leq} 0.53^2 \cdot f(i) \cdot n.$$

Bounding p requires slightly more complex arguments. For $3 \leq i \leq \ell$, the probability that a fixed ball of batch t is allocated to height $t + i$ is at most $f(i - 1)^d$. This is because each of its locations has to point to one of the bins with $t + i - 1$ or more balls, and by our induction on i , the fraction of these bins is bounded from above by $f(i - 1)$. Taking into account all n balls of batch t , we obtain

$$\mathbf{E}[p] \leq f(i - 1)^d \cdot n \stackrel{(A3)}{\leq} 0.7 \cdot f(i) \cdot n.$$

Applying a Chernoff bound yields, for every $\epsilon \in (0, 1]$,

$$\begin{aligned} \Pr[p \geq (1 + \epsilon) \cdot 0.7 \cdot f(i) \cdot n] &\leq \exp\left(\frac{-0.7 \epsilon^2}{2} \cdot f(i) \cdot n\right) \\ &\stackrel{(A4)}{\leq} \exp\left(\frac{-0.7 \epsilon^2}{8} \cdot n^{0.1}\right) \\ &\leq n^{-\kappa}, \end{aligned}$$

where the last inequality holds for any given κ and $\epsilon > 0$, provided n is sufficiently large. Hence, $p \leq (1 + \epsilon) \cdot 0.7 \cdot f(i) \cdot n$, w.h.p. Consequently, $\beta_i^{(t)} \cdot n \leq q + p \leq (0.53^2 + 0.7 \cdot (1 + \epsilon)) \cdot f(i) \cdot n$, w.h.p. We set $\epsilon = 0.02$ so that $(0.53^2 + 0.7 \cdot (1 + \epsilon)) \leq 1$. This proves invariant $H_1(t)$ for $2 \leq i \leq \ell$.

Finally, we prove invariant $H_2(t)$. For $0 \leq \tau \leq t$, let x_τ denote a random variable which is one if at least one ball of batch τ is allocated into a bin with load larger than $\tau + \ell$, and zero, otherwise. Furthermore, let h_τ denote the number of balls from batch τ that are allocated into a bin with load larger than $\tau + \ell$. Because of the invariants $H_1(1), \dots, H_1(t)$, the probability that a fixed ball from batch τ will fall into a bin with more than $\tau + \ell$ balls is at most $f(\ell)^d \leq (n^{-0.9})^d \leq n^{-1.8}$. Therefore, $\Pr[x_\tau = 1] \leq n \cdot n^{-1.8} = n^{-0.8}$. In particular, for any integer $j \geq 1$,

$$\Pr[h_\tau \geq j] \leq \binom{n}{j} \left(\frac{1}{n^{1.8}}\right)^j \leq n^{-0.8j}.$$

In other words, $h_\tau = O(1)$, w.h.p. Thus, we can assume that there exists a suitable constant j so that $h_\tau \leq j$ for $1 \leq \tau \leq t$. A violation of $H_2(t)$ implies that the bins with load at least $t + \ell + 1$ contain more than b balls of height at least $t + \ell + 1$. Observe that these balls must have been placed during the last b rounds or one of the invariants $H_2(1), \dots, H_2(t-1)$ is violated. That is, if $H_2(1), \dots, H_2(t-1)$ hold, then a violation of $H_2(t)$ implies that $j \cdot \sum_{\tau=t-b}^t x_\tau \geq b$. The probability for this event is at most

$$\Pr \left[\sum_{\tau=t-b}^t x_\tau \geq b/j \right] \leq \binom{b}{b/j} \cdot \left(\frac{1}{n^{0.8}} \right)^{b/j} \leq \left(\frac{e j}{n^{0.8}} \right)^{b/j} \leq n^{-\kappa}$$

for any constant κ , provided that n is sufficiently large. Consequently, invariant H_2 holds, w.h.p., over all batches. This completes the proof of Lemma 1.1.

3. Greedy has short memory. The goal of this section is to prove the Short Memory Lemma, Lemma 1.2. We will study the performance of Greedy[d] by analyzing the underlying Markov chain describing the load distribution of the bins and our analysis uses a new variant of coupling approach (neighboring coupling (see section 3.1.2)) to study the convergence time of Markov chains.

Remark 1. It is well known that the Short Memory Lemma does not hold for $d = 1$, that is, for the single-choice algorithm. In that case, in order to claim that \mathcal{X}_τ has stochastically almost the same distribution as \mathcal{Y}_τ one must have $\tau = \Omega(m^2 \cdot \text{poly}(n))$.

3.1. Auxiliary lemmas. In this subsection we state some auxiliary results that will be used in order to prove Lemma 1.2.

3.1.1. A simple load estimation. We present here the following simple (and known) lemma (see, e.g., [15, 17, 30]) that we use in the proof of Lemma 3.9.

LEMMA 3.1. *Suppose that m balls are allocated in n bins using Greedy[2]. Let p be any positive real. Then with probability at least $1 - p$ the minimum load in any bin is larger than or equal to*

$$\frac{m}{n} - \sqrt{\frac{2m}{n} \cdot \ln \frac{n}{p}}.$$

Proof. The proof follows easily from the fact that the minimum load in Greedy[2] is (stochastically) not smaller than the minimum load in the process that allocates m balls in n bins i.u.r. (this follows, for example, from the majorization results presented in [1, Theorem 3.5]). The minimum load in the process that allocates m balls in n bins i.u.r. can be estimated by looking at each bin independently. Then, we apply the Chernoff bound² to estimate the probability that the load of the bin is smaller than the expected load (i.e., from $\frac{m}{n}$) by more than $\sqrt{\frac{2m}{n} \cdot \ln \frac{n}{p}}$. We apply the Chernoff bound to m independent random variables X_1, \dots, X_m with X_i indicating the event that the i th ball is allocated in the given bin. Then, we obtain for every $t \in \mathbb{N}$

$$\Pr \left[\text{given bin has load} \leq \frac{m}{n} - t \right] \leq \exp \left(\frac{-n t^2}{2m} \right).$$

²We use the following form of the Chernoff bound (see, e.g., [21, Theorem 2.3 (c)]): If X_1, \dots, X_m are binary independent random variables and if $X = \sum_{j=1}^m X_j$, then for any δ , $0 < \delta < 1$, it holds that $\Pr[X \leq (1 - \delta) \mathbf{E}[X]] \leq \exp(-\delta^2 \mathbf{E}[X]/2)$.

Therefore, by the union bound,

$$\Pr \left[\text{minimum load} \leq \frac{m}{n} - t \right] \leq n \cdot \exp \left(\frac{-nt^2}{2m} \right).$$

This implies our result by setting $t = \sqrt{\frac{2m}{n} \cdot \ln \frac{n}{p}}$. \square

3.1.2. Neighboring coupling. Our main tool in the analysis of the convergence of Markov chains is a new variant of the path coupling arguments that extends the results from [7, 13] and which is described in the following lemma.

LEMMA 3.2 (Neighboring Coupling Lemma). *Let $\mathfrak{M} = (\mathbf{Y}_t)_{t \in \mathbb{N}}$ be a discrete-time Markov chain with a state space Ω . Let $\Omega^* \subseteq \Omega$. Let Γ be any subset of $\Omega^* \times \Omega^*$ (elements $(X, Y) \in \Gamma$ are called neighbors). Suppose that there is an integer D such that for every $(X, Y) \in \Omega^* \times \Omega^*$ there exists a sequence $X = \Lambda_0, \Lambda_1, \dots, \Lambda_r = Y$, where $(\Lambda_i, \Lambda_{i+1}) \in \Gamma$ for $0 \leq i < r$, and $r \leq D$.*

If there exists a coupling $(X_t, Y_t)_{t \in \mathbb{N}}$ for \mathfrak{M} such that for some $T \in \mathbb{N}$, for all $(X, Y) \in \Gamma$, it holds that $\Pr[X_T \neq Y_T \mid (X_0, Y_0) = (X, Y)] \leq \frac{\varepsilon}{D}$, then

$$\|\mathcal{L}(X_T \mid X_0 = X) - \mathcal{L}(Y_T \mid Y_0 = Y)\| \leq \varepsilon$$

for every $(X, Y) \in \Omega^* \times \Omega^*$.

Proof. For any pair of neighbors $(\Lambda, \Lambda') \in \Gamma$,

$$\|\mathcal{L}(Z_T \mid Z_0 = \Lambda) - \mathcal{L}(Z_T \mid Z_0 = \Lambda')\| \leq \frac{\varepsilon}{D}$$

by the well-known Coupling Lemma (see, e.g., [4, Lemma 3.6]). As a consequence, we obtain

$$\begin{aligned} & \|\mathcal{L}(Z_T \mid Z_0 = X) - \mathcal{L}(Z_T \mid Z_0 = Y)\| \\ & \leq \sum_{i=1}^r \|\mathcal{L}(Z_T \mid Z_0 = \Lambda_i) - \mathcal{L}(Z_T \mid Z_0 = \Lambda_{i-1})\| \leq r \cdot \frac{\varepsilon}{D} \leq \varepsilon. \quad \square \end{aligned}$$

Thus, if we can find a neighboring coupling, we obtain immediately a bound on the total variation distance in terms of the tail probabilities of the coupling time, i.e., a random time \mathbb{T} for which $X_t = Y_t$ for all $t \geq \mathbb{T}$.

3.1.3. Random walks on \mathbb{N} . In this section we present some auxiliary results on the convergence rates of a random walk on the line \mathbb{N} with the starting point D , with the absorbing barrier at 0, and with a drift $\beta \geq 0$ toward 0. We feel that these results might be known, but since we have not seen them in forms needed in our paper, we decided to present them here in detail for the sake of completeness.

We begin with the following result which can be viewed as a bound on the number of steps needed by a random walk on the integer line with positive drift toward 0 until the “barrier” 0 is hit.

LEMMA 3.3 (random walk on \mathbb{N} —positive drift toward 0). *Let ϵ and β be any positive reals. Let D be an arbitrary natural number. Let $c \geq 1$ be an arbitrary constant. Let $(\mathcal{X}_t)_{t \in \mathbb{N}}$ be a sequence of (not necessarily independent) random variables such that*

- (1) $0 \leq \mathcal{X}_0 \leq D$,
- (2) for every $t \in \mathbb{N}$, $|\mathcal{X}_{t+1} - \mathcal{X}_t| \leq c$,
- (3) for every $t \in \mathbb{N}$, if $\mathcal{X}_t > 0$ then $\mathbf{E}[\mathcal{X}_{t+1} - \mathcal{X}_t \mid \mathcal{X}_0, \mathcal{X}_1, \dots, \mathcal{X}_t] \leq -\beta$, and

(4) for every $t \in \mathbb{N}$, if $\mathcal{X}_t \leq 0$ then $\mathcal{X}_{t+1} = 0$.

Then, for certain $T = \Theta(D/\beta + \ln(1/\epsilon)/\beta^2)$, if $\tau \geq T$ then $\Pr[\mathcal{X}_\tau > 0] \leq \epsilon$.

Proof. We eliminate the requirement that \mathcal{X}_t is never negative by introducing a new sequence of random variables $(\mathcal{Y}_t)_{t \in \mathbb{N}}$. If $\mathcal{X}_t > 0$ then we set $\mathcal{Y}_t = \mathcal{X}_t$, and otherwise we define $\mathcal{Y}_t = \mathcal{Y}_{t-1} - \beta$.

Then, conditions (1)–(2) still hold for sequence $(\mathcal{Y}_t)_{t \in \mathbb{N}}$, condition (3) holds for $(\mathcal{Y}_t)_{t \in \mathbb{N}}$ even without the assumption that $\mathcal{Y}_t > 0$, and we have a new condition (4*) such that for every $t \in \mathbb{N}$, if $\mathcal{Y}_t \leq 0$ then $\mathcal{Y}_{t+1} \leq 0$. Additionally, it is easy to see that for every $t \in \mathbb{N}$ it holds that $\Pr[\mathcal{Y}_t > 0] = \Pr[\mathcal{X}_t > 0]$.

Next, since for every t we have $\mathbf{E}[\mathcal{Y}_{t+1} - \mathcal{Y}_t \mid \mathcal{Y}_0, \dots, \mathcal{Y}_t] \leq -\beta$, we see that $\mathbf{E}[\mathcal{Y}_t] \leq D - \beta t$. Therefore, we can apply the Hoeffding–Azuma inequality (see, e.g., [21, Theorem 3.10]) to obtain that for every $\alpha > 0$ it holds that

$$\Pr[\mathcal{Y}_t > \alpha + \mathbf{E}[\mathcal{Y}_t]] \leq e^{-\alpha^2/(2tc)}.$$

Since for $\alpha = t\beta - D$ we have $\alpha + \mathbf{E}[\mathcal{Y}_t] \leq 0$, we can conclude that

$$\Pr[\mathcal{X}_t > 0] = \Pr[\mathcal{Y}_t > 0] \leq e^{-(t\beta - D)^2/(2tc)},$$

which yields the lemma. \square

Next, we investigate random walks on the integers with “balanced” drift, or a drift which is very small; that is, β in Lemma 3.3 is tiny and hence the bound at that lemma is weak.

LEMMA 3.4 (random walk on \mathbb{N} —balanced case). *Let ϵ be any positive real. Let D be an arbitrary natural number. Let $c \geq 1$ be an arbitrary constant. Let $(\mathcal{X}_t)_{t \in \mathbb{N}}$ be a sequence of (not necessarily independent) integer random variables such that the following properties hold:*

1. $0 \leq \mathcal{X}_t \leq D$ for every $t \in \mathbb{N}$,
2. for every $t \in \mathbb{N}$, $|\mathcal{X}_{t+1} - \mathcal{X}_t| \leq c$,
3. for every $t \in \mathbb{N}$, if $\mathcal{X}_t > 0$ then $\mathcal{X}_{t+1} \neq \mathcal{X}_t$ and $\mathbf{E}[\mathcal{X}_{t+1} - \mathcal{X}_t \mid \mathcal{X}_0, \mathcal{X}_1, \dots, \mathcal{X}_t] \leq 0$, and
4. for every $t \in \mathbb{N}$, if $\mathcal{X}_t = 0$ then $\mathcal{X}_{t+1} = 0$.

Then, for certain $T = \Theta(D^2 \cdot \ln(1/\epsilon))$, if $\tau \geq T$ then $\Pr[\mathcal{X}_\tau > 0] \leq \epsilon$.

Proof. We first observe that it is enough to prove the lemma only for $\mathbf{E}[\mathcal{X}_{t+1} - \mathcal{X}_t \mid \mathcal{X}_0, \dots, \mathcal{X}_t] = 0$. We follow here arguments given in [19, Lemma 4]. Recall that random variables $\mathcal{V}_0, \mathcal{V}_1, \dots$, form a *submartingale* with respect to a sequence $(\mathcal{W}_t)_{t \in \mathbb{N}}$ if $\mathbf{E}[\mathcal{V}_{t+1} - \mathcal{V}_t \mid \mathcal{W}_0, \mathcal{W}_1, \dots, \mathcal{W}_t] \geq 0$ for every $t \in \mathbb{N}$. A random variable τ is a *stopping time* for the submartingale if for each t one can determine if $\tau \geq t$. We shall use the optional stopping time theorem (due to Doob) for submartingales which says that if $(\mathcal{U}_t)_{t \in \mathbb{N}}$ is a submartingale $(\mathcal{U}_t)_{t \in \mathbb{N}}$ with bounded $|\mathcal{U}_{t+1} - \mathcal{U}_t|$ for every $t \in \mathbb{N}$ and τ is a stopping time with finite expectation, then $\mathbf{E}[\mathcal{U}_\tau] \geq \mathbf{E}[\mathcal{U}_0]$.

Fix \mathcal{X}_0 . Define the stochastic process $\mathcal{Z}_0, \mathcal{Z}_1, \dots$ such that for every $t \in \mathbb{N}$,

$$\mathcal{Z}_t = \begin{cases} (D - \mathcal{X}_t)^2 - t & \text{if either } t = 0 \text{ or } t > 0 \text{ and } \mathcal{X}_{t-1} > 0, \\ \mathcal{Z}_{t-1} & \text{if } t > 0 \text{ and } \mathcal{X}_{t-1} = 0. \end{cases}$$

Let us first observe that \mathcal{Z}_t is a submartingale with respect to the sequence $(\mathcal{X}_t)_{t \in \mathbb{N}}$. Indeed, conditioned on $\mathcal{X}_0, \mathcal{X}_1, \dots, \mathcal{X}_t$, if $\mathcal{X}_t = 0$ then $\mathcal{Z}_{t+1} - \mathcal{Z}_t = 0$. Otherwise, if $\mathcal{X}_t > 0$, then we obtain $\mathbf{E}[\mathcal{Z}_{t+1} - \mathcal{Z}_t] = \mathbf{E}[(D - \mathcal{X}_{t+1})^2 - (t + 1) - ((D - \mathcal{X}_t)^2 - t)] = \mathbf{E}[(\mathcal{X}_{t+1} - \mathcal{X}_t)^2 - 1 + 2 \cdot (\mathcal{X}_t - \mathcal{X}_{t+1}) \cdot (D - \mathcal{X}_t)] \geq \mathbf{E}[(\mathcal{X}_{t+1} - \mathcal{X}_t)^2] - 1 + 2 \cdot (D - \mathcal{X}_t) \cdot \mathbf{E}[\mathcal{X}_t -$

$\mathcal{X}_{t+1}] \geq 1 - 1 + 0 \geq 0$. Here, we used the fact that if $\mathcal{X}_t > 0$ then $\mathcal{X}_{t+1} \neq \mathcal{X}_t$, and therefore since each of \mathcal{X}_{t+1} and \mathcal{X}_t is an integer, it holds that $\mathbf{E}[(\mathcal{X}_{t+1} - \mathcal{X}_t)^2] \geq 1$.

Next, we notice that the differences $|\mathcal{Z}_{t+1} - \mathcal{Z}_t|$ are bounded for every $t \in \mathbb{N}$ and the random time $\mathfrak{T}_{\mathcal{X}_0} = \min\{t : \mathcal{X}_t = 0\}$ is a stopping time for \mathcal{Z}_t with finite expectation. Moreover, $\mathcal{Z}_0 = (D - \mathcal{X}_0)^2$ and $\mathcal{Z}_{\mathfrak{T}_{\mathcal{X}_0}} = D^2 - \mathfrak{T}_{\mathcal{X}_0}$. Since $\mathbf{E}[\mathcal{Z}_{\mathfrak{T}_{\mathcal{X}_0}}] \geq \mathbf{E}[\mathcal{Z}_0]$ by the optional stopping time theorem, we get $\mathbf{E}[\mathfrak{T}_{\mathcal{X}_0}] \leq \mathcal{X}_0 \cdot (2D - \mathcal{X}_0) \leq D^2$.

Take $t = e \cdot D^2$. The Markov inequality implies that $\Pr[\mathfrak{T}_{\mathcal{X}_0} \geq t] \leq e^{-1}$. If we run $\ln(1/\epsilon)$ independent trials of length t then for $T = t \cdot \ln(1/\epsilon)$ the probability that $\mathcal{X}_T \neq 0$ is at most ϵ^{-1} . \square

3.2. Greedy[d] has short memory: Analysis for $d=2$. In this section we prove Lemma 1.2 for $d = 2$. More precisely, we will prove various properties about Greedy[d]. For $d = 2$ these properties will immediately imply Lemma 1.2. For $d > 2$ we need some additional arguments which are then presented in section 3.3.

Throughout this section we deal only with *normalized* load vectors. For every $k \geq 0$, let Ω_k denote the set of normalized load vectors with k balls. In our analysis, we investigate the following Markov chain $\mathfrak{M}[d] = (\mathcal{M}_t)_{t \in \mathbb{N}}$, which models the behavior of protocol Greedy[d]:

Input: \mathcal{M}_0 is any load vector in Ω_m

Transitions $\mathcal{M}_t \Rightarrow \mathcal{M}_{t+1}$:

Pick $q \in [n]$ at random such that $\Pr[q = k] = \frac{k^d - (k-1)^d}{n^d}$

\mathcal{M}_{t+1} is obtained from \mathcal{M}_t by adding a new ball to the q th fullest bin

It is easy to see that the choice of q is equivalent to the choice obtained by the following simple randomized process: Pick $q_1, q_2, \dots, q_d \in [n]$ i.u.r. and set $q = \max\{q_i : 1 \leq i \leq d\}$. This in turn, is equivalent to the choice of q obtained by Greedy[d]: Pick d bins i.u.r. and let the least loaded among the chosen bins be the q th fullest bin in the system.

Our proof of Lemma 1.2 is via the *neighboring coupling* method discussed in detail in section 3.1.2. Let X and Y denote two vectors from Ω_m . We study the process by which we add new balls on the top of each of the allocations described by these vectors. We analyze how many balls one has to add until the two allocations are almost indistinguishable.

3.2.1. Neighboring coupling. In order to apply the Neighboring Coupling Lemma to analyze the Markov chain $\mathfrak{M}[d] = (\mathcal{M}_t)_{t \in \mathbb{N}}$, we must first define the notion of neighbors. Let us fix m and n . Let us define $\Omega^* = \Omega_m$ and let Γ be the set of pairs of those load vectors from Ω_m which correspond to the balls' allocations that differ in exactly one ball (cf. Figure 2). In that case, if X can be obtained from Y by moving a ball from the i th fullest bin into the j th fullest bin, then we write $X = Y - \mathbf{e}_i + \mathbf{e}_j$. Thus,

$$\Gamma = \{(X, Y) \in \Omega_m \times \Omega_m : X = Y - \mathbf{e}_i + \mathbf{e}_j \text{ for certain } i, j \in [n], i \neq j\}.$$

Clearly, for each $X, Y \in \Omega_m$ there exists a sequence $X = Z^{(0)}, Z^{(1)}, \dots, Z^{(l-1)}, Z^{(l)} = Y$, where l is the number of balls on which X and Y differ, $l \leq m$, and $(Z^{(i)}, Z^{(i+1)}) \in \Gamma$ for every i , $0 \leq i \leq l - 1$. Thus, we can apply the Neighboring Coupling Lemma with $D = m$.

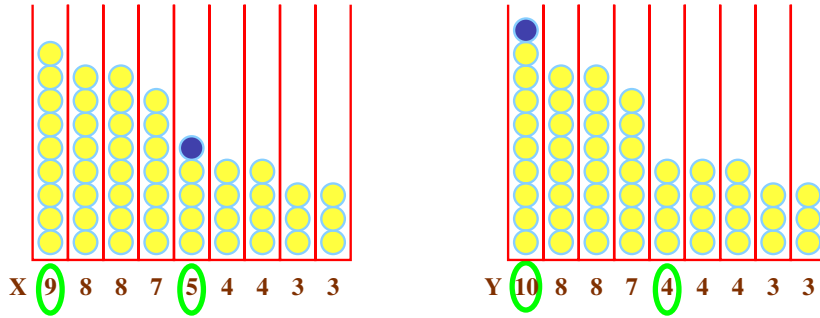


FIG. 2. An example of neighboring load vectors X and Y with $X = Y - \mathbf{e}_1 + \mathbf{e}_5$ and $\Delta(X, Y) = 6$.

3.2.2. Short memory lemma using neighboring coupling. The main result of this section is the following technical lemma.

LEMMA 3.5. *Let $\varepsilon > 0$ and let $d \geq 2$ be integer. Let $X, Y \in \Omega_m$. There exists a coupling $(X_t, Y_t)_{t \in \mathbb{N}}$ for $\mathfrak{M}[d]$ and there is $\mathbb{T} = \Theta(m \cdot n^d + n^{2d} \cdot \ln(m/\varepsilon))$ such that for any $\tau \geq \mathbb{T}$ it holds that $\Pr[X_\tau \neq Y_\tau \mid (X_0, Y_0) = (X, Y)] \leq \varepsilon$.*

Let us first observe that for $d = 2$ Lemma 3.5 immediately implies the Short Memory Lemma for Greedy[2]. However, Lemma 3.5 yields a weaker bound for $d \geq 3$. Therefore, a more specialized analysis for the case $d \geq 3$ is postponed to section 3.3.

Notice that since for any pair $X, Y \in \Omega_m$ there exists a sequence $\Lambda_0, \Lambda_1, \dots, \Lambda_k$ such that $k \leq m$, $\Lambda_0 = X$, $\Lambda_k = Y$, and $(\Lambda_{i-1}, \Lambda_i) \in \Gamma$ for every $1 \leq i \leq k$, Lemma 3.5 follows immediately from the following lemma.

LEMMA 3.6. *Let $\varepsilon > 0$ and let $d \geq 2$ be integer. Let $X, Y \in \Gamma$. There exists a coupling $(X_t, Y_t)_{t \in \mathbb{N}}$ for $\mathfrak{M}[d]$ and there is $\mathbb{T} = \Theta(m \cdot n^d + n^{2d} \cdot \ln(m/\varepsilon))$ such that for any $\tau \geq \mathbb{T}$ it holds that $\Pr[X_\tau \neq Y_\tau \mid (X_0, Y_0) = (X, Y)] \leq \frac{\varepsilon}{m}$.*

The rest of this subsection is devoted to the proof of Lemma 3.6.

For any load vectors $X = (x_1, \dots, x_n)$ and $Y = (y_1, \dots, y_n)$ with $X = Y - \mathbf{e}_i + \mathbf{e}_j$, $i, j \in [n]$, let us define the *distance* function $\Delta(X, Y)$ (cf. Figure 2),

$$\Delta(X, Y) = \max\{|x_i - x_j|, |y_i - y_j|\}.$$

Observe that $\Delta(X, Y)$ is always a nonnegative integer, it is zero only if $X = Y$ and it never takes the value of 1. Let

$$\begin{aligned} \xi &= \min\{\Pr[\text{Greedy}[d] \text{ picks the } j\text{th fullest bin}] \\ &\quad - \Pr[\text{Greedy}[d] \text{ picks the } i\text{th fullest bin}] : i, j \in [n], i < j\}. \end{aligned}$$

Then, clearly, $\xi \geq 1/n^d$. The following lemma describes main properties of the desired coupling.

LEMMA 3.7. *If $(X, Y) \in \Gamma$ then there exists a coupling $(X_t, Y_t)_{t \in \mathbb{N}}$ for $\mathfrak{M}[d]$ that, conditioned on $(X_0, Y_0) = (X, Y)$, satisfies the following properties for every $t \in \mathbb{N}$:*

- if $X_t = Y_t$ then $X_{t+1} = Y_{t+1}$,
- if $X_t \neq Y_t$ then X_t and Y_t differ in at most one ball,
- $\Delta(X_{t+1}, Y_{t+1}) - \Delta(X_t, Y_t) \in \{-2, -1, 0, 1\}$, and
- if $X_t \neq Y_t$ then $\mathbf{E}[\Delta(X_{t+1}, Y_{t+1}) \mid X_t, Y_t] \leq \Delta(X_t, Y_t) - \xi$.

Proof. We use the following natural coupling: each time we increase the vectors X and Y by one ball, we use the same random choice. That is, in each step the obtained load vectors will be obtained from X and Y , respectively, by allocating a new ball to the q th fullest bin for certain $q \in [n]$.

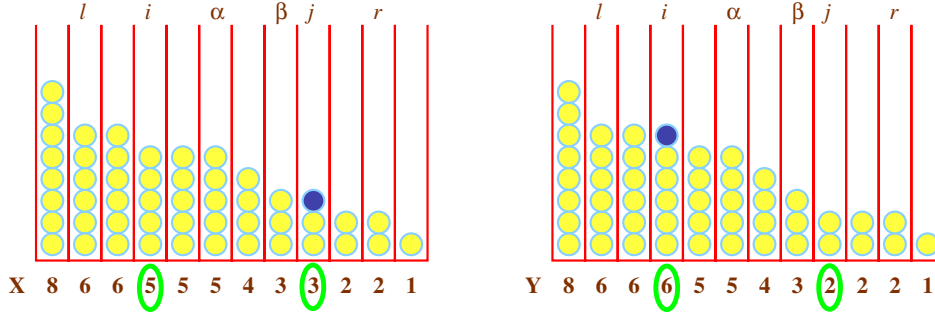


FIG. 3. Illustration to the proof of Claim 3.8. In this case $X = Y - e_4 + e_9$, $\Delta(X, Y) = 4$, $l = 2$, $\alpha = 6$, $\beta = 8$, and $r = 11$.

The lemma now follows directly from the properties of the coupling described in Claim 3.8 below. \square

CLAIM 3.8. Let X, Y be two load vectors from Ω_m that differ in one ball with $X = Y - e_i + e_j$ for certain $i < j$. Let $X^{(q)}$ and $Y^{(q)}$ be obtained from X and Y , respectively, by allocating a new ball to the q th fullest bin. Then, either

1. $X^{(q)} = Y^{(q)}$ and $\Delta(X^{(q)}, Y^{(q)}) = \Delta(X, Y) - 2$, or
2. $X^{(q)}$ and $Y^{(q)}$ differ in one ball and

$$\Delta(X^{(q)}, Y^{(q)}) = \begin{cases} \Delta(X, Y) - 1 & \text{if and only if } q = j, \\ \Delta(X, Y) + 1 & \text{if and only if } q = i, \\ \Delta(X, Y) & \text{otherwise.} \end{cases}$$

Proof. The proof is by case analysis which is tedious but otherwise straightforward (see also Figure 3 for some intuition behind the coupling). We assume that $\Delta(X, Y) = y_i - y_j$; the case $\Delta(X, Y) = x_i - x_j$ can be done similarly. Let

$$\begin{aligned} l &= \min\{s \in [n] : y_s = y_i\}, \\ \alpha &= \max\{s \in [n] : x_i = x_s\}, \\ \beta &= \min\{s \in [n] : x_s = x_j\}, \\ r &= \max\{s \in [n] : y_j = x_s\}. \end{aligned}$$

Let us notice that $1 \leq l \leq i \leq \alpha$, $l < \alpha$, $\beta \leq j \leq r \leq n$, and $\beta < r$. We first consider six cases when either $1 \leq q \leq i$ or $j \leq q \leq n$.

Case (1) $1 \leq q < l$. Since $q < l$ the same happens for both processes. For certain s , $s \leq q$, we have $X^{(q)} = X + e_s$ and $Y^{(q)} = Y + e_s$. Therefore, after adding the ball to the q th fullest bin, we still have $X^{(q)} = Y^{(q)} - e_i + e_j$. Hence, $\Delta(X^{(q)}, Y^{(q)}) = y_i^{(q)} - y_j^{(q)} = y_i - y_j = \Delta(X, Y)$.

Case (2) $l \leq q < i$. After reordering of the bins, the load of the l th largest bin has increased by one for both load vectors (note that all load vectors between the l th largest and the $i - 1$ th largest bin have the same load in both processes). Hence, $X^{(q)} = X + e_l$ and $Y^{(q)} = Y + e_l$. Therefore, $X^{(q)} = Y^{(q)} - e_i + e_j$ and the rest of the case is similar to Case (1).

Case (3) $q = i$. In this case we have $X^{(q)} = X + e_i$. After reordering of the load vector of Y we have $Y^{(q)} = Y + e_l$ (see Case (2)). This yields $X^{(q)} = Y^{(q)} - e_l + e_j$ and $\Delta(X^{(q)}, Y^{(q)}) = y_l^{(q)} - y_j^{(q)} = (y_l + 1) - y_j = \Delta(X, Y) + 1$.

Case (4) $q = j$. The β th and the j th largest bins have the same number of elements after adding a ball to the j th largest bin of X . Hence, $X^{(q)} = X + e_\beta$ and

$Y^{(q)} = Y + \mathbf{e}_j$ (in the case of Y the bins do not have to be reordered). Therefore $X^{(q)} = Y^{(q)} - \mathbf{e}_i + \mathbf{e}_\beta$.

Now, there are two possibilities. First, if $\Delta(X, Y) = 2$, then $i = \beta$ and therefore $X^{(q)} = Y^{(q)}$. Otherwise, $\Delta(X, Y) > 2$ and hence $i \leq \alpha < \beta$, which implies that $\Delta(X^{(q)}, Y^{(q)}) = y_i^{(q)} - y_\beta^{(q)} = y_i - (y_j + 1) = \Delta(X, Y) - 1$.

Case (5) $j < q \leq r$. After reordering we have $X^{(q)} = X + \mathbf{e}_{j+1}$ and $Y^{(q)} = Y + \mathbf{e}_j$. We get $X^{(q)} = Y^{(q)} - \mathbf{e}_i + \mathbf{e}_{j+1}$ and $\Delta(X^{(q)}, Y^{(q)}) = y_i^{(q)} - y_{j+1}^{(q)} = y_i - y_j = \Delta(X, Y)$.

Case (6) $r < q \leq n$. This case is similar to Case (1). For certain $s, r < s \leq q$, it holds that $X^{(q)} = X + \mathbf{e}_s$ and $Y^{(q)} = Y + \mathbf{e}_s$. Therefore, $X^{(q)} = Y^{(q)} - \mathbf{e}_i + \mathbf{e}_j$ and $\Delta(X^{(q)}, Y^{(q)})$ does not change.

Now it remains to consider the case when $i < q < j$. We distinguish here two main cases.

Case (A) $x_i = x_j$. In this case we have $\Delta(X, Y) = 2$. After reordering we have $X^{(q)} = X + \mathbf{e}_i$ and $Y^{(q)} = Y + \mathbf{e}_{i+1}$. This means $X^{(q)} = Y^{(q)} - \mathbf{e}_{i+1} + \mathbf{e}_j$ and $\Delta(X^{(q)}, Y^{(q)}) = y_{i+1}^{(q)} - y_j^{(q)} = y_i - y_j = 2 = \Delta(X, Y)$.

Case (B) $x_i > x_j$. In this case $\alpha < \beta$ and we distinguish three subcases:

Case (B.1) $i < q \leq \alpha$. After reordering we get $X^{(q)} = X + \mathbf{e}_i$ and $Y^{(q)} = Y + \mathbf{e}_{i+1}$. Therefore, $X^{(q)} = Y^{(q)} - \mathbf{e}_{i+1} + \mathbf{e}_j$ and $\Delta(X^{(q)}, Y^{(q)}) = y_{i+1}^{(q)} - y_j^{(q)} = y_i - y_j = \Delta(X, Y)$.

Case (B.2) $\alpha < q < \beta$. Again, this case is similar to Case (1); for certain $s, \alpha < s \leq q < \beta$, we get $X^{(q)} = X + \mathbf{e}_s$ and $Y^{(q)} = Y + \mathbf{e}_s$. Hence, $\Delta(X, Y)$ does not change.

Case (B.3) $\beta \leq q < j$. In this case $X^{(q)} = X + \mathbf{e}_\beta$ and $Y^{(q)} = Y + \mathbf{e}_\beta$. Therefore, $X^{(q)} = Y^{(q)} - \mathbf{e}_i + \mathbf{e}_j$ and $\Delta(X, Y)$ does not change. \square

Now we are ready to present the proof of Lemma 3.6.

Proof. We use the coupling constructed in Lemma 3.7. Observe that if we define $\Delta_t = \Delta(X_t, Y_t), t \geq 0$, then from Lemma 3.7 the random variable Δ_t behaves like a random walk on \mathbb{N} with drift toward 0; see section 3.1.3. By our assumption, we can set $\xi = 1/n^d$. Given that, we can conclude the proof by applying Lemma 3.3 with $\mathcal{X}_t = \Delta(X_t, Y_t), c_t = 2$ for every $t \in \mathbb{N}$, and with $D = m$ and $\beta = \xi = 1/n^d$. \square

3.3. Short memory property of Greedy[d] for large d . The main problem with applying Lemma 3.7 for large d is that the value of ξ may be very small. Now we modify the analysis above to give a better bound for Greedy[d] than the one of Lemma 3.5 for all $d > 2$.

3.3.1. Load difference reduction in Greedy[d] for $d \geq 3$. For any load vector \mathcal{W} let us denote by $\text{LOW}(\mathcal{W})$ ($\text{UPP}(\mathcal{W})$) the minimum load (respectively, the maximum load) in \mathcal{W} . We prove that independently of the initial difference between $\text{LOW}(\mathcal{W})$ and $\text{UPP}(\mathcal{W})$ at some moment of the allocation process Greedy[d], after allocating some new balls, this difference will be kept small.

LEMMA 3.9. *Let n and M be any positive integers and let ε be any positive real. Let $d \geq 2$ be any integer. Let $X \in \Omega_M$. Let $\mathfrak{M}[d] = (X_t)_{t \in \mathbb{N}}$ with $X_0 = X$; that is, X_0, X_1, \dots is the sequence of random variables describing the Markov chain $\mathfrak{M}[d]$ conditioned on the event $X_0 = X$. Then, there exist a certain $\mathbb{T} = \Theta(Mn^2 + n^4 \cdot \ln(M/\varepsilon))$ and a constant $c > 0$ such that the following hold.*

(1) If $M = \mathcal{O}(n^3 \ln(n/\varepsilon))$, then the following two bounds hold:

$$\Pr \left[\text{LOW}(X_{\mathbb{T}}) \leq \frac{M + \mathbb{T}}{n} - c \cdot \sqrt{(M + n^2 \ln(M/\varepsilon)) \cdot n \cdot \ln(n/\varepsilon)} \right] \leq \varepsilon,$$

$$\Pr \left[\text{UPP}(X_{\mathbb{T}}) \geq \frac{M + \mathbb{T}}{n} + c \cdot \sqrt{(M + n^2 \ln(M/\varepsilon)) \cdot n \cdot \ln(n/\varepsilon)} \right] \leq \varepsilon.$$

(2) If $M = \Omega(n^3 \ln(n/\varepsilon))$, then the following two bounds hold:

$$\Pr \left[\text{LOW}(X_{\mathbb{T}}) \leq \frac{M + \mathbb{T}}{n} - c \cdot n^{1.25} \cdot M^{0.25} \cdot (\ln(n/\varepsilon))^{0.75} \right] \leq \varepsilon,$$

$$\Pr \left[\text{UPP}(X_{\mathbb{T}}) \geq \frac{M + \mathbb{T}}{n} + c \cdot n^{1.25} \cdot M^{0.25} \cdot (\ln(n/\varepsilon))^{0.75} \right] \leq \varepsilon.$$

Proof. We prove the lemma only for M being the multiple of n ; the general case can be handled similarly. Since the proofs for $\text{LOW}(\mathcal{W})$ and $\text{UPP}(\mathcal{W})$ are almost the same, we will deal only with $\text{LOW}(\mathcal{W})$. We also point out that our proof uses ideas similar to those discussed later in section 4.

Let Y_0, Y_1, \dots be the sequence of random variables (normalized load vectors) describing the Markov chain $\mathfrak{M}[2]$ conditioned on the event $Y_0 = X$. It is known that for every $l \in \mathbb{N}$ the normalized load vector X_l is majorized by the normalized load vector Y_l (see, e.g., [1, Theorem 3.5]). Therefore, in particular, $\text{LOW}(X_l)$ is stochastically larger than or equal to $\text{LOW}(Y_l)$ (the minimum load in Y_l). Hence, it is enough to prove the lemma only for the load vectors Y_0, Y_1, \dots .

Let ς be any positive real. Let Z be the ideally balanced load vector in Ω_M (i.e., the loads of all bins in Z are the same). Let Z_0, Z_1, \dots be the sequence of random variables (normalized load vectors) describing the Markov chain $\mathfrak{M}[2]$ conditioned on the event $Z_0 = Z$. Lemma 3.5 implies that for certain $T = \Theta(M n^2 + n^4 \cdot \ln(M/\varsigma))$, for any $t \geq T$ the load vectors Y_t and Z_t are almost indistinguishable (formally, $\|\mathcal{L}(Z_t) - \mathcal{L}(Y_t)\| \leq \varsigma$). In particular, this means that the random variables $\text{LOW}(Y_t)$ and $\text{LOW}(Z_t)$ are stochastically the same with probability at least $1 - \varsigma$. Furthermore, by Lemma 3.1, we know that for any $t \in \mathbb{N}$, it holds that

$$(1) \quad \Pr \left[\text{LOW}(Z_t) \leq \frac{M + t}{n} - \sqrt{\frac{2(M + t)}{n} \ln \frac{n}{\varsigma}} \right] \leq \varsigma.$$

Therefore, since for any $t \geq T$ we have $\|\mathcal{L}(Z_t) - \mathcal{L}(Y_t)\| \leq \varsigma$, we may conclude that for $t \geq T$ it holds that

$$(2) \quad \Pr \left[\text{LOW}(Y_t) \leq \frac{M + t}{n} - \sqrt{\frac{2(M + t)}{n} \ln \frac{n}{\varsigma}} \right] \leq 2\varsigma.$$

With inequality (2) we immediately obtain the first estimation for LOW by setting $\varepsilon = 2\varsigma$ and $\mathbb{T} = t = T$.

In order to obtain the second estimation, we first fix the smallest $\tau \geq T$ such that τ is a multiple of n . Let \mathcal{E} be the event that $\text{LOW}(Y_\tau) \geq \frac{M + \tau}{n} - \sqrt{\frac{2(M + \tau)}{n} \ln \frac{n}{\varsigma}}$. Let us condition on this event for a moment.

For any $t \geq \tau$, let Y_t^* be the load vector obtained from Y_t after removing $r = \lfloor \frac{M + \tau}{n} - \sqrt{\frac{2(M + \tau)}{n} \ln \frac{n}{\varsigma}} \rfloor$ balls from each bin in Y_t . Clearly, since $\text{LOW}(Y_t) \geq$

$\text{Low}(Y_\tau) \geq r$, Y_t^* is a proper normalized load vector in $\Omega_{t+M-r \cdot n}$. Notice further that there are $M^* = M + \tau - r \cdot n$ balls in the system described by Y_τ^* .

Now we apply once again a similar procedure as we did above for the first estimation. Let V be the ideally balanced load vector in Ω_{M^*} (i.e., each bin in V has the same load). Let V_0, V_1, \dots be the sequence of random variables (normalized load vectors) describing the Markov chain $\mathfrak{M}[2]$ conditioned on the event $V_0 = V$. Proceeding similarly as above, we want to compare $Y_{t+\tau}^*$ with V_t for $t \geq 0$.

Lemma 3.5 implies that for certain $T^* = \Theta(M^* n^2 + n^4 \cdot \ln(M^*/\varsigma))$, for any $t \geq T^*$ it holds that $\|\mathcal{L}(V_t) - \mathcal{L}(Y_{t+\tau}^*)\| \leq \varsigma$. Therefore, in particular, the random variables $\text{Low}(Y_{t+\tau}^*)$ and $\text{Low}(V_t)$ are stochastically the same with probability at least $1 - \varsigma$. Furthermore, by Lemma 3.1, we know that for any $t \in \mathbb{N}$ it holds that

$$\Pr \left[\text{Low}(V_t) \leq \frac{M^* + t}{n} - \sqrt{\frac{2(M^* + t)}{n} \ln \frac{n}{\varsigma}} \right] \leq \varsigma.$$

Therefore, since $\|\mathcal{L}(V_t) - \mathcal{L}(Y_{t+\tau}^*)\| \leq \varsigma$ for any $t \geq T^*$, we may conclude that for any $t \geq T^*$ it holds that

$$\Pr \left[\text{Low}(Y_{t+\tau}^*) \leq \frac{M^* + t}{n} - \sqrt{\frac{2(M^* + t)}{n} \ln \frac{n}{\varsigma}} \right] \leq 2\varsigma.$$

Furthermore, since the load vector $Y_{t+\tau}^*$ is obtained from the load vector $Y_{t+\tau}$ by removing r balls from each bin, we obtain that (conditioned on \mathcal{E}) for any $t \geq T^*$,

$$\Pr \left[\text{Low}(Y_{t+\tau}) \leq \frac{M + \tau + t}{n} - \sqrt{\frac{2(M^* + t)}{n} \ln \frac{n}{\varsigma}} \mid \mathcal{E} \right] \leq 2\varsigma.$$

Finally, since we have proved that event \mathcal{E} holds with probability at least $1 - 2\varsigma$ (see inequality (2)), we can conclude that for any $t \geq T^*$ it holds that

$$\Pr \left[\text{Low}(Y_{t+\tau}) \leq \frac{M + \tau + t}{n} - \sqrt{\frac{2(M^* + t)}{n} \ln \frac{n}{\varsigma}} \right] \leq 4\varsigma.$$

Now, it remains to resolve this bound with respect to n , M , and ς . We observe that

$$\tau = \Theta(M n^2 + n^4 \ln(M/\varsigma))$$

and

$$M^* = \Theta \left(\sqrt{\tau n \ln(n/\varsigma)} \right) = \Theta \left(n^{1.5} \sqrt{M \ln(n/\varsigma)} + n^{2.5} \sqrt{\ln(M/\varsigma) \ln(n/\varsigma)} \right).$$

Furthermore,

$$\begin{aligned} T^* &= \Theta(M^* n^2 + n^4 \cdot \ln(M^*/\varsigma)) \\ &= \Theta \left(n^{3.5} \sqrt{M \cdot \ln(n/\varsigma)} \quad n^{4.5} \sqrt{\ln(M/\varsigma) \ln(n/\varsigma)} \right. \\ &\quad \left. + n^4 \ln \left(\frac{n M \ln(n/\varsigma) \ln(M/\varsigma)}{\varsigma} \right) \right). \end{aligned}$$

Now, we use our assumption that $M = \Omega(n^3 \ln(n/\varsigma))$ for $\varsigma = \Theta(\varepsilon)$. In this case, the first term dominates the other one in the bounds for τ and for M^* and the first

term dominates the other two in the bound for T^* . Hence, $\tau = \Theta(Mn^2)$, $M^* = \Theta(n^{1.5} \sqrt{M \ln(n/\zeta)})$, $T^* = \Theta(n^{3.5} \sqrt{M \cdot \ln(n/\zeta)})$, and $\tau + T^* = \Theta(\tau)$. Therefore, we can conclude with the following claim: There exists a positive constant c such that for $\mathbb{T} = \tau + T^*$ it holds that

$$\Pr \left[\text{Low}(X_{\mathbb{T}}) \leq \frac{M + \mathbb{T}}{n} - c \cdot n^{1.25} \cdot M^{0.25} \cdot (\ln(n/\zeta))^{0.75} \right] \leq 4\zeta.$$

Now, the lemma follows by setting $\varepsilon = 4\zeta$. \square

3.3.2. Coupling arguments and the short memory lemma for $d > 2$.

Observe that a trivial implication of Lemma 3.9 is that if we start with any pair of normalized load vectors $X_0, Y_0 \in \Omega_M$, then for certain $\tau = \Theta(Mn^2 + n^4 \ln(M/\varepsilon))$, w.h.p. (depending on ε and κ), it holds that for any integer $\kappa > 0$, and for all t , $0 \leq t < \kappa$, the difference in the maximum load and the minimum load in $X_{\tau+t}$ (or $Y_{\tau+t}$) is upper-bounded by

$$\zeta = \begin{cases} \mathcal{O} \left(\sqrt{(M + n^2 \ln(M/\varepsilon)) n \ln(n/\varepsilon)} \right) & \text{for } M = \mathcal{O}(n^3 \ln(n/\varepsilon)), \\ \mathcal{O} \left(n^{1.25} M^{0.25} (\ln(n/\varepsilon))^{0.75} \right) & \text{for } M = \Omega(n^3 \ln(n/\varepsilon)). \end{cases}$$

From now on we shall fix τ and κ and shall condition on this event (which, by applying the union bound to Lemma 3.9, is satisfied with probability larger than $1 - 2\kappa\varepsilon$).

Now we proceed with coupling arguments. Let $X, Y \in \Gamma$. We use the same distance function $\Delta(\cdot, \cdot)$ as in section 3 and the same coupling as in Lemma 3.7 and Claim 3.8. Observe that by our discussion in section 3.2.2, for any $t \in \mathbb{N}$, either X_t and Y_t are identical or they differ by one ball.

Epochs. Let τ and κ be set as above. We divide the time into *epochs*. The 0th epoch starts at time step $l_0 = 0$ and ends in time step $r_0 = \tau$. Each following epoch corresponds to the time period when the value of Δ remains unchanged. That is, if the $(k - 1)$ st epoch, $k \geq 1$, ends in time step r_{k-1} , then, inductively, the k th epoch begins in time step $l_k = 1 + r_{k-1}$ and ends in the smallest time step $t \geq l_k$ for which $\Delta(X_{t-1}, Y_{t-1}) \neq \Delta(X_t, Y_t)$. Additionally, if $X_{r_{k-1}} = Y_{r_{k-1}}$, then we define $r_k = \infty$, and the k th epoch lasts until the infinity.

CLAIM 3.10. *Let μ be any positive integer. Let for every t , $\tau \leq t \leq \tau + \kappa$, the difference between the maximum load and the minimum load in each of X_t and Y_t be upper-bounded by μ . Then, for every $1 \leq k \leq \frac{\kappa}{2n\mu + 1}$, if $X_{r_{k-1}} \neq Y_{r_{k-1}}$, then the k th epoch lasts at most $2n\mu + 1$ time steps.*

Proof. Let $X_t = Y_t - \mathbf{e}_i + \mathbf{e}_j$ with $i < j$ and let $X_{t+1} = Y_{t+1} - \mathbf{e}_{i^*} + \mathbf{e}_{j^*}$, where X_{t+1} and Y_{t+1} are obtained from X_t and Y_t , respectively, by allocating a ball to bin q . By the case analysis (cf. also the proof of Claim 3.8) one can show that one of the following two cases must hold:

- $q = i$ or $q = j$ in the transition $(X_t, Y_t) \rightarrow (X_{t+1}, Y_{t+1})$ of the coupling.
- The load of the i^* th fullest bin in Y_{t+1} is the same as the load of the i th fullest bin in Y_t .

Consider a k th epoch and suppose that $X_{r_{k-1}} \neq Y_{r_{k-1}}$ with $X_{r_{k-1}} = Y_{r_{k-1}} - \mathbf{e}_{i^*} + \mathbf{e}_{j^*}$, $i^* < j^*$, in time r_{k-1} . Let ℓ be the load of the i^* th fullest bin in $Y_{r_{k-1}}$. Then, by the observation above and by Claim 3.8, for every t with $r_{k-1} \leq t \leq r_k - 1$, if $X_t = Y_t - \mathbf{e}_i + \mathbf{e}_j$ with $i < j$, then the load of the i th fullest bin in Y_t is ℓ .

Now, we want to use the assumption that for every t , $\tau \leq t \leq \tau + \kappa$, the difference between the maximum load and the minimum load in Y_t is upper-bounded by μ . Therefore, if $\tau \leq r_{k-1} \leq \tau + \kappa$, then in time r_{k-1} the value of ℓ (which is the load of

one of the bins in Y_{r_k-1}) is at most $\frac{r_k-1}{n} + \mu$. Furthermore, if $\tau \leq r_k \leq \tau + \kappa$, then in time $r_k - 1$ the value of ℓ (which is the load of one of the bins in Y_{r_k-1}) is at least $\frac{r_k-1}{n} - \mu$. Therefore, we obtain that

$$\frac{r_k-1}{n} + \mu \geq \ell \geq \frac{r_k-1}{n} - \mu.$$

This implies immediately that $r_k - l_k \leq 2n\mu$. This also yields inductively that if $1 \leq k \leq \kappa/(2n\mu + 1)$ and $X_{r_{k-1}} \neq Y_{r_{k-1}}$, then the k th epoch lasts at most $2n\mu + 1$ time steps. \square

Let Δ_k be the value of $\Delta(X_t, Y_t)$ for $t = r_k$, i.e., for t being the last time step of the k th epoch. Clearly, $\Delta_{k-1} \neq \Delta_k$. Furthermore, by Claim 3.8 the following holds:

- If $\Delta_{k-1} \geq 3$ then $\Delta_k \in \{\Delta_{k-1} - 1, \Delta_{k-1} + 1\}$ and $\mathbf{E}[\Delta_k - \Delta_{k-1}] < 0$.
- If $\Delta_{k-1} = 2$ then $\Delta_k \in \{0, 3\}$ and $\mathbf{E}[\Delta_k - \Delta_{k-1}] < 0$.

Therefore, similarly as in the proof of Lemma 3.5, we can model sequence $\Delta_0, \Delta_1, \dots$ as a random walk on the line \mathbb{N} with the starting point $D \leq \zeta$, the absorbing barrier in 0, and with a positive drift toward 0. This time, however, the drift is very small and therefore we use a weaker bound for the convergence of this random walk. From Lemma 3.4 we obtain that

$$\mathbf{Pr}[\Delta_k > 0] \leq 2\kappa\varepsilon \quad \text{for all } k \geq \gamma\zeta^2 \ln(2\kappa\varepsilon)^{-1},$$

where γ is some absolute positive constant. Now, since by Claim 3.10 each epoch k with $\Delta_{k-1} > 0$ lasts at most $2n\zeta + 1$ time steps, the last inequality implies the following. For all $t, t \geq \tau + \gamma\zeta^2 \ln(2\kappa\varepsilon)^{-1} \cdot (2n\zeta + 1)$, it holds that $\mathbf{Pr}[X_t \neq Y_t] \leq 2\kappa\varepsilon$. Since, by Lemma 3.9, we have proven that with probability larger than or equal to $1 - 2\kappa\varepsilon$, for every $t, \tau \leq t < \tau + \kappa$, the difference between the maximum load and the minimum load in each of X_t and Y_t is upper bounded by ζ , we can conclude with the following lemma.

LEMMA 3.11. *Let n and m be any positive integers and let ε be any positive real. Let $d \geq 2$ be any integer. Let $X_0, Y_0 \in \Omega_m$.*

- (1) *If $m = \mathcal{O}(n^3 \ln(n/\varepsilon))$, then there is a coupling $(X_t, Y_t)_{t \in \mathbb{N}}$ for $\mathfrak{M}[d]$ such that for certain \mathbb{T}^* ,*

$$\mathbb{T}^* = \Theta(n^{2.5} \cdot (m + n^2 \ln(n/\varepsilon))^{1.5} \cdot \ln(n/\varepsilon)^{1.5} \cdot \ln(1/\varepsilon));$$

it holds for any $t \geq \mathbb{T}^$ that*

$$\mathbf{Pr}[X_t \neq Y_t \mid (X_0, Y_0) = (X, Y)] \leq \varepsilon.$$

- (2) *If $m = \Omega(n^3 \ln(n/\varepsilon))$, then there is a coupling $(X_t, Y_t)_{t \in \mathbb{N}}$ for $\mathfrak{M}[d]$ such that for certain \mathbb{T}^* ,*

$$\mathbb{T}^* = \Theta(m \cdot n^2 + m^{0.75} \cdot n^{4.75} \cdot (\ln(m/\varepsilon))^{2.25} \cdot \ln(1/\varepsilon));$$

it holds for any $t \geq \mathbb{T}^$ that*

$$\mathbf{Pr}[X_t \neq Y_t \mid (X_0, Y_0) = (X, Y)] \leq \varepsilon.$$

Now, Lemma 3.11 directly implies the Short Memory Lemma, Lemma 1.2, for all values of d .

4. A reduction to a polynomial number of balls for Greedy[d]. In this section, we discuss our main use of the Short Memory Lemma which is a reduction of the analysis of the problem of allocating an arbitrary number m of balls into n bins to the case when m is upper-bounded by a polynomial of n . If we combine this analysis with the analysis for a polynomial number of balls in section 2, we will immediately obtain Theorem 1.3.

Our arguments are similar to those used in section 3.3. We begin with the following corollary which follows directly from Lemma 1.2.

COROLLARY 4.1. *Let X_0 be any normalized load vector describing an arbitrary allocation of some number m of balls to n bins. Let Δ be the difference between the maximum and the minimum load in X_0 . Let Y_0 be the normalized load vector describing the optimally balanced allocation of m balls into n bins (that is, each bin in Y_0 has either $\lfloor m/n \rfloor$ or $\lceil m/n \rceil$ balls). Let X_k and Y_k , respectively, denote the vectors obtained after inserting $k \geq 1$ further balls into both systems using Greedy[d]. For every constant α there is a constant c such that if $k \geq cn^7 \Delta \ln^4(n\Delta)$, then*

$$\|\mathcal{L}(X_k) - \mathcal{L}(Y_k)\| \leq k^{-\alpha}.$$

Proof. Let ℓ denote the minimum load of any bin in X_0 . We consider the scenario after removing ℓ balls from each bin in X_0 and Y_0 ; let X_0^* and Y_0^* be the respective load vectors. X_0^* and Y_0^* have an identical number of balls that we denote by m^* . Observe that since the maximum load in X_0 was $\ell + \Delta$, we have $m^* \leq n\Delta$.

Next, let X_t^* and Y_t^* , respectively, denote the vectors obtained after inserting $t \geq 1$ further balls to the systems corresponding to X_0^* and Y_0^* , where we use Greedy[d] to place the balls. We apply the Short Memory Lemma, Lemma 1.2, to the sequences X_t^* and Y_t^* to obtain that there is $\tau \leq c'm^* n^6 \ln^4(1/\varepsilon) \leq c'\Delta n^7 \ln^4(1/\varepsilon)$ for a suitable constant $c' > 0$, such that for every $t \geq \tau$ we have $\|\mathcal{L}(X_t^*) - \mathcal{L}(Y_t^*)\| \leq \varepsilon$. Therefore, if we set $\varepsilon = k^{-\alpha}$ and choose k such that it satisfies $k = c'\Delta n^7 \alpha \ln^4 k = O(n^7 \Delta \ln^4(n\Delta))$, we have $\|\mathcal{L}(X_k^*) - \mathcal{L}(Y_k^*)\| \leq k^{-\alpha}$.

Now, the claim follows from the fact that for every $t \geq 0$ the distributions of X_t and Y_t^* , and Y_t and Y_t^* , respectively, differ only in that every bin corresponding to X_t (Y_t) has ℓ less balls than the corresponding bin in X_t^* (Y_t^* , respectively). \square

Using Corollary 4.1, we present a general transformation which shows that the allocation obtained by an allocation process with “short memory” is more or less independent of the number of balls. The following theorem shows that the allocation (its distribution) is essentially determined after inserting a polynomial number of balls. In particular, this theorem together with Lemma 1.1 immediately imply Theorem 1.3. We assume that n is sufficiently large.

THEOREM 4.2. *Let us consider the process Greedy[d], $d \geq 2$, in which the balls are allocated into n bins. For any integer m , let $X_m = (x_1^{(m)}, \dots, x_n^{(m)})$ be a load vector obtained after allocating m balls with Greedy[d] and let $\tilde{X}_m = (x_1^{(m)} - \frac{m}{n}, \dots, x_n^{(m)} - \frac{m}{n})$. Let $N = n^{36}$. Then, for every M , $M \geq N$, that is a multiple of n ,*

$$\|\mathcal{L}(\tilde{X}_M) - \mathcal{L}(\tilde{X}_N)\| \leq N^{-\alpha},$$

where α denotes an arbitrary constant.

Proof. Let us first begin with the claim that if M and m are multiples of n with $M \geq n^{36}$ and $M \geq m \geq M^{0.8}$, then

$$(3) \quad \|\mathcal{L}(\tilde{X}_M) - \mathcal{L}(\tilde{X}_m)\| \leq M^{-\alpha}.$$

Let us set $m' = M - m$. We use the majorization from the single-choice process to estimate the distribution of the bins' loads after inserting m' balls. Since Greedy[d] is majorized by the single-choice process, each bin contains $\frac{m'}{n} \pm \mathcal{O}(\sqrt{m' \ln(n/p)/n})$ balls, with probability at least $1 - p$ for any $p \in [0, 1]$. We set $\Delta = M^{0.6}$ and $p = M^{-\alpha}/2$. Applying $M \geq m' \geq n$ yields that every bin contains between $\frac{m'}{n} - \Delta/2$ and $\frac{m'}{n} + \Delta/2$ balls, with probability at least $1 - p$, provided that n is sufficiently large. Let us now condition on this event and assume that the entries in $\tilde{X}_{m'}$ are in the Δ -range specified above.

Let Y describe another system in which the first m' balls are inserted in an optimal way; that is, $Y_{m'} = (\frac{m'}{n}, \dots, \frac{m'}{n})$. Now, we add m balls using protocol Greedy[d] on top of $X_{m'}$ and $Y_{m'}$, respectively. Now, applying Corollary 4.1, we obtain $\|\mathcal{L}(X_M) - \mathcal{L}(Y_M)\| \leq m^{-2\alpha} \leq M^{-\alpha}/2$ as $m \geq M^{0.8} \geq n^7 M^{0.6} \ln^4 M \geq c n^7 \Delta \ln^4(n \Delta)$, where c is the constant specified in the corollary. Thus, conditioned on the event that the values in $\tilde{X}_{m'}$ are in the interval $[\frac{m'}{n} - \Delta/2, \frac{m'}{n} + \Delta/2]$, we have $\mathcal{L}(\tilde{Y}_M) = \mathcal{L}(\tilde{X}_m)$, with probability at least $1 - M^{-\alpha}/2$. Therefore, since the condition is satisfied with probability at least $1 - M^{-\alpha}/2$ as well, we have

$$\|\mathcal{L}(\tilde{X}_M) - \mathcal{L}(\tilde{X}_m)\| \leq M^{-\alpha}/2 + \|\mathcal{L}(X_M) - \mathcal{L}(Y_M)\| \leq M^{-\alpha}/2 + M^{-\alpha}/2 = M^{-\alpha},$$

which completes the proof of inequality (3).

Finally, we use inequality (3) to prove Theorem 4.2. Observe first that if $M \leq N^{1/0.8}$, then (3) directly implies Theorem 4.2. Otherwise, we have to apply inequality (3) repeatedly as follows. Let m_0, m_1, \dots, m_k denote a sequence of integers such that $m_0 = N$, $m_k = M$, $m_i^{0.8} \leq m_{i-1}$, and $m_i^\alpha \geq 2m_{i-1}^\alpha$. Then

$$\|\mathcal{L}(\tilde{X}_M) - \mathcal{L}(\tilde{X}_N)\| \leq \sum_{i=1}^k \|\mathcal{L}(\tilde{X}_{m_i}) - \mathcal{L}(\tilde{X}_{m_{i-1}})\| \leq \sum_{i=1}^k m_i^{-\alpha} \leq N^{-\alpha},$$

where the last inequality follows from the fact that $m_i^{-\alpha} \leq 2^{-i} m_0^{-\alpha} = 2^{-i} N^{-\alpha}$. \square

Remark 2. It is easy to see that the proof above does not use any of the properties of Greedy[d] but the following two: Corollary 4.1 and the fact that Greedy[d] is majorized by the single-choice process. Therefore, Theorem 4.2 holds for any allocation protocol \mathcal{P} that has short memory (in the sense of Corollary 4.1) and that is majorized by the single-choice process.

5. Greedy[d] majorizes Left[d]. In this section, we will begin our analysis of the always-go-left allocation scheme and prove Theorem 1.7. Let $d \geq 2$, n be any multiple of d , and $m \geq 0$. We show that Left[d] is majorized by Greedy[d].

Our proof is by induction on the number of balls in the system. Let u denote the load vector obtained after inserting some number of balls with Left[d], and let v denote the load vector obtained after inserting the same number of balls with Greedy[d]. Without loss of generality, we assume that u and v are normalized, i.e., $u_1 \geq u_2 \geq \dots \geq u_n$ and $v_1 \geq v_2 \geq \dots \geq v_n$. Notice that the normalization of u jumbles the bins in the different groups used by Left[d] in some unspecified way so that it remains unclear which bin belongs to which group. Let u' and v' denote the load vectors obtained by adding another ball b with Left[d] and Greedy[d], respectively. To prove Theorem 1.7 by induction, we show that if $u \leq v$ then there is a coupling of Left[d] and Greedy[d] with respect to the allocation of b such that $u' \leq v'$, regardless of the unspecified mapping of the bins to the groups underlying u .

As a first step in the description of the coupling, we replace the original formulations of the allocation rules of the two random processes by alternative formulations that enable us to define an appropriate coupling.

- At first, we describe the alternative formulation of the allocation rules for Greedy[d]. For $1 \leq i \leq n$, let \mathbf{e}_i denote the i th unit vector, and define $b_i = \Pr[v' = v + \mathbf{e}_i]$. For $0 \leq i \leq n$, let $B_i = \sum_{j=1}^i b_j$; that is, B_i denotes the probability that the next ball is added to a bin with index at most i with respect to the considered order of bins. Without loss of generality, we assume that Greedy[d] gives the ball to the bin with larger index in case of a tie. Then $B_i = (i/n)^d$ because Greedy[d] places the ball b in a bin with index smaller than or equal to i if and only if all of the d locations of b point to bins whose indices are at most i . Instead of inserting the next ball using the rules of Greedy[d], we now choose a continuous random variable x uniformly at random from the interval $[0, 1]$ and allocate b in the i th bin if $B_{i-1} < x \leq B_i$. By our construction, this results in the same distribution.
- Now we turn our attention to Left[d]. Given any allocation of the balls to bins corresponding to the load vector u , define $a_i = \Pr[u' = u + \mathbf{e}_i]$ for $1 \leq i \leq n$ and $A_i = \sum_{j=1}^i a_j$ for $0 \leq i \leq n$. Observe that the probabilities a_i and A_i do not depend only on the index i (as in the case of Greedy[d]) or the vector u , but also on the hidden mapping of the bins to the groups. Consequently, we cannot specify these terms as a functional of i or u . Nevertheless, for any given mapping of the bins to groups, the terms A_0, \dots, A_n are well defined so that we can replace the original allocation rules by the following rule that results in the same distribution: Choose a random variable x uniformly at random from the interval $[0, 1]$ and allocate the ball b into the i th bin if $A_{i-1} < x \leq A_i$.

For the coupling, we now assume that Left[d] and Greedy[d] use the same random number x to assign the ball b . By our construction, this coupling is faithful. Under the coupling, we have to show $u' \leq v'$. Let $u' = u + \mathbf{e}_i$ and $v' = v + \mathbf{e}_j$ for some i and j ; that is, i and j specify the indices of the bins in the vectors u and v into which Left[d] and Greedy[d], respectively, put the ball b .

First, let us assume that the initial vectors u and v are equal. In this case, we have to show that $u + \mathbf{e}_i \leq u + \mathbf{e}_j$. Consider the plateaus of u , i.e., maximal index sets of bins with the same height. Suppose there are $k \geq 2$ plateaus U_1, \dots, U_k such that the load is decreasing from U_1 to U_k . Let I and J denote the indices of the plateaus that contain i and j , respectively. Observe that $I \geq J$ implies $u + \mathbf{e}_i \leq u + \mathbf{e}_j$ because adding a ball to different positions of the same plateau results in the same normalized vector. Thus, we have only to show that $J \leq I$. Let $\ell = \max\{U_I\}$. Since $i \leq \ell$, we have $x \leq A_\ell$. In the following lemma, we show that $A_\ell \leq (\ell/n)^d$. Above we have shown that $B_\ell = (\ell/n)^d$. Therefore, the lemma implies $x \leq B_\ell$, which shows that Greedy[d] places its ball in a bin with index at most ℓ ; that is, $j \leq \ell$ and, hence, $J \leq I$. Consequently, $u + \mathbf{e}_i \leq u + \mathbf{e}_j$.

LEMMA 5.1. *For any mapping of the bins to the groups underlying the vector u , $A_\ell \leq (\ell/n)^d$.*

Proof. Recall that A_ℓ corresponds to the probability that Left[d] places the ball b in a location with index (with respect to u) smaller than or equal to ℓ . Let ℓ_k for $0 \leq k < d$ denote the number of bins in group k with load greater than or equal to u_ℓ . Then $A_\ell = \prod_{k=0}^{d-1} \frac{\ell_k}{n/d}$ because the k th location of b must be one of those ℓ_k bins among the d/n bins in group k that have load at least u_ℓ . Since $\sum_{k=0}^{d-1} \ell_k = \ell$, A_ℓ is

maximized for $\ell_0 = \dots = \ell_{d-1} = \ell/d$. Consequently, $A_\ell \leq (\ell/n)^d$. \square

Until now we have analyzed only the case when $u = v$ and have shown $u + \mathbf{e}_i \leq u + \mathbf{e}_j$. This, however, can be easily generalized to arbitrary normalized load vectors. Indeed, for any two normalized load vectors w and w' , $w \leq w'$ implies $w + \mathbf{e}_i \leq w' + \mathbf{e}_i$, cf. [1, Lemma 3.4]. Consequently, we can conclude from $u + \mathbf{e}_i \leq u + \mathbf{e}_j$ that $u' = u + \mathbf{e}_i \leq u + \mathbf{e}_j \leq v + \mathbf{e}_j = v'$. Thus, Theorem 1.7 is shown.

6. Analysis of Left[d]. In this section, we investigate the allocation generated by Left[d]. In particular, we prove Theorem 1.5, that is, we show that the number of bins with load at least $\frac{m}{n} + i + \gamma$ is at most $n \cdot \exp(-\phi_d^{d \cdot i})$, w.h.p., where ϕ_d denotes the d -ary golden ratio (cf. section 1.1) and γ is a suitable constant. Similarly to the proof for Greedy[d], we divide the set of balls into batches of size n and we apply an induction on the number of batches. On one hand, the proof for Left[d] is slightly more complicated since we have to take into account that the set of bins is partitioned into d groups. On the other hand, we can avoid the detour through analyzing the holes below average height as we can instead make use of the majorization of Left[d] by Greedy[d].

For the time being, let us assume that $m \geq n \log_2 n$. We will use the majorization from Greedy[d] to estimate the allocation after allocating the first $m' = m - n \log_2 n$ balls. The special properties of Left[d] will only be taken into account for the remaining $n \log_2 n$ balls. Let us divide the set of these balls into $\log_2 n$ batches of size n each. Let time 0 denote the point of time before the first ball from batch 1 is inserted, that is, after inserting the first m' balls; and, for $1 \leq t \leq \log_2 n$, let time t denote the point of time after inserting the balls from batch t . Furthermore, set $\Gamma = \frac{m'}{n} + 7$ and, for $i \geq 0, 0 \leq j < d, 0 \leq t \leq \log_2 n$, let $\nu_{i,j}^{(t)}$ denote the number of bins with load at least $\Gamma + t + i$ in group j at time t . The following lemma gives an upper bound on the allocation of Left[d] obtained by the majorization from Greedy[d] at time 0. This upper bound is specified in terms of the function

$$h_0(i) = \frac{1}{4^i \cdot 64d}.$$

Later we will use the same lemma to estimate parts of the allocation also for other points of time, $t \geq 1$.

LEMMA 6.1. *Let ℓ denote the smallest integer such that $h_0(\ell) \leq n^{-0.9}$, i.e., $\ell = \lfloor 0.9 \log_4 n - \log_4 d \rfloor - 2$. For $0 \leq i < \ell, 0 \leq j < d, 0 \leq t \leq \log_2 n$, it holds $\nu_{i,j}^{(t)} \leq h_0(i) \cdot n/d$, w.h.p. For $i \geq \ell, \nu_{i,j}^{(t)} = 0$, w.h.p.*

Proof. Fix a time step t . Theorem 1.3 shows that, when using Greedy[d], the fraction of bins with load at least $\frac{m'}{n} + t + i$ is upper-bounded by a function that decreases doubly exponentially in i . Now, in order to simplify the subsequent calculations, we upper-bound this function by another function that decreases only exponentially in i , namely, the function h_0 . With lots of room to spare, the analysis in section 2.3 yields that the fraction of bins with load at least $\Gamma + t + i$ can be upper-bounded by $h_0(i)/(2d)$, w.h.p., provided n is sufficiently large. This result holds for Greedy[d], and we want to apply it to Left[d] via majorization. In order to make use of the majorization of Left[d] by Greedy[d], we need a bound on the number of balls above some given height rather than a bound on the number of bins with load above the height. However, since the bound given above on the number of bins decreases geometrically in i , the number of balls of height at least $\Gamma + t + i$ when using Greedy[d] is bounded from above by $h_0(i) \cdot n/d$. Now, because of the majorization, this result holds for Left[d], too. In turn, the number of balls above height $\Gamma + t + i$ upper-bounds

the number of bins with load at least $\Gamma + t + i$. Hence, when using $\text{Left}[d]$, the total number of bins with load at least $\Gamma + t + i$ is bounded from above by $h_0(i) \cdot n/d$. Of course, the same upper bound holds for the number of such bins in each individual group.

Finally, it remains to be shown that $\nu_{i,j}^{(t)} = 0$, w.h.p., for $i \geq \ell$ with $\ell = \lfloor 0.9 \log_4 n - \log_4 d \rfloor - 2$. Again this follows via majorization from $\text{Greedy}[d]$. Theorem 1.3 implies that the maximum load of $\text{Greedy}[d]$ and, hence, also of $\text{Left}[d]$ at time t is bounded from above by $\Gamma + t + O(\log_d \log n)$, w.h.p. Thus there is no ball with height $\Gamma + t + \ell$, w.h.p. \square

Before we turn to the technical details, let us explain the high-level idea behind the following analysis. We will use a function $f(k, t)$ as an upper bound for $\nu_{\lfloor k/d \rfloor, j \bmod d}^{(t)}$. For $t = 0$, $f(k, t)$ will be set equal to $h_0(\lfloor k/d \rfloor)$, and we will use the above lemma to show that $f(k, 0)$ upper-bounds $\nu_{\lfloor k/d \rfloor, k \bmod d}^{(0)}$. When increasing t , the function $f(k, t)$ will become more similar to the function $h_1(k)$ defined by

$$h_1(k) = \frac{\exp(-F_d(k - d + 1))}{64d},$$

where $F_d(k)$ denotes the k th d -ary Fibonacci number as defined in section 1.1. Let $i = \lfloor k/d \rfloor$ and $j = k \bmod d$. Then $h_1(k)$ will serve as an upper bound on the fraction of bins with height $\Gamma + t + i$ in group j . As explained in section 1.1, we use the d -ary golden ratio to upper-bound the d -ary Fibonacci numbers. This way,

$$h_1(k) = \frac{\exp(-\phi_d^{k \pm O(d)})}{64d} = \frac{\exp(-\phi_d^{(i \pm O(1))d})}{64d}.$$

Hence, for large t , the fraction of bins with some given height decreases ‘‘Fibonacci exponentially’’ with the height, exactly as described in Theorem 1.5.

Now we come to the technical details. We define

$$f(k, t) = \max\{h_0(\lfloor k/d \rfloor) \cdot 2^{-t}, h_1(k)\}.$$

Observe that f changes smoothly from h_0 into h_1 when increasing t . In particular, $f(k, 0) = h_0(\lfloor k/d \rfloor)$ and $f(k, \log_2 n) \leq h_1(k) + \frac{1}{n}$. We need to refine the function f slightly. Intuitively, we truncate the function when the function values become ‘‘too small.’’ Let c denote a sufficiently large constant term, whose value will be specified later. Let ℓ_t denote the smallest integer such that $f(\ell_t, t) \leq n^{-0.9}$. We set

$$f'(k, t) = \begin{cases} \max\left\{\frac{n^{-0.9}}{4}, f(k, t)\right\} & \text{if } 0 \leq k < \ell_t + d, \\ \frac{cd}{n} & \text{if } k \geq \ell_t + d. \end{cases}$$

The following properties of f' are crucial for our analysis. They hold only if n is sufficiently large.

LEMMA 6.2.

- B1. $f'(k, t) = h_0(0)$ for $0 \leq k < d, t \geq 0$;
- B2. $f'(k, t) \geq 2 \cdot f'(k + d, t - 1)$ for $d \leq k < \ell_t + d, t \geq 1$;
- B3. $f'(k, t) \geq (4d) \cdot \prod_{j=1}^d f'(k - j, t)$ for $d \leq k < \ell_t + d, t \geq 0$;
- B4. $f'(k, t) \geq n^{-0.9}/4$ for $d \leq k < \ell_t, t \geq 0$.

Proof. We start with the proof of property B1. First, let us check the property for f instead of f' . For $0 \leq k < d$, $F_d(k - d + 1) = 0$ so that $h_1(k) = 1/(64d) = h_0(0)$. Hence, for every $t \geq 0$,

$$f(k, t) = \max\{h_0(\lfloor k/d \rfloor) \cdot 2^{-t}, h_1(k)\} = h_0(0).$$

If n is sufficiently large then the same is true for f' .

Next we show property B2, first for f and then for f' . The function $f(k, t)$ is defined by the maximum of the two terms $h_0(\lfloor k/d \rfloor) \cdot 2^{-t}$ and $h_1(k)$. We study these terms one after the other. The definition of h_0 immediately implies

$$h_0(\lfloor k/d \rfloor) \cdot 2^{-t} = 4h_0(\lfloor (k + d)/d \rfloor) \cdot 2^{-t} = 2h_0(\lfloor (k + d)/d \rfloor) \cdot 2^{-(t-1)}.$$

Furthermore, for $k \geq d$,

$$h_1(k) = \frac{\exp(-F_d(k - d + 1))}{64d} \geq \frac{2 \exp(-F_d(k + 1))}{64d} = 2h_1(k + d).$$

As a consequence, $f(k, t) \geq 2f(k + d, t - 1)$, that is, B2 is shown for f . The refinement from f to f' might raise the right-hand side of the inequality from $2f(k + d, t - 1)$ to the value $2n^{-0.9}/4$, or the right-hand side might take the value $2cd/n$. At first, suppose $f'(k + d, t - 1) = n^{-0.9}/4$. Then $k < \ell_{t-1}$ so that $f(k, t - 1) \geq n^{-0.9}$. Now this implies $f(k, t) \geq n^{-0.9}/2$ as $f(k, t) \geq f(k, t - 1)/2$. Consequently,

$$f'(k, t) = \max \left\{ f(k, t), \frac{n^{-0.9}}{4} \right\} \geq \frac{n^{-0.9}}{2} = 2f'(k + d, t - 1).$$

In the second case, $f'(k + d, t - 1) = cd/n$. Observe that property B2 needs to be shown only for $k < \ell_t + d$. For this choice of k , $f'(k, t) \geq n^{-0.9}/4$ so that $f'(k, t) \geq 2f'(k + d, t - 1)$ if n is sufficiently large. Hence, B2 is shown.

Property B3 is shown as follows. Again we first show the property for f and then for f' . Fix $k \geq d$. Depending on the outcome of the terms $f(k - d, t), \dots, f(k - 1, t)$, we distinguish two cases. First, suppose there exists $\delta \in \{1, \dots, d\}$ such that $f(k - \delta, t) = h_0(\lfloor (k - \delta)/d \rfloor) \cdot 2^{-t}$. Observe that $h_0(\lfloor (k - \delta)/d \rfloor) \leq h_0(\lfloor k/d \rfloor - 1) = 4 h_0(\lfloor k/d \rfloor)$. We obtain

$$\begin{aligned} \prod_{j=1}^d f(k - j, t) &= h_0(\lfloor (k - \delta)/d \rfloor) \cdot 2^{-t} \cdot \prod_{\substack{j=1 \\ j \neq \delta}}^d f(k - j, t) \\ &\leq 4 h_0(\lfloor k/d \rfloor) \cdot 2^{-t} \cdot \left(\frac{1}{64d}\right)^{d-1} \\ &\leq \frac{h_0(\lfloor k/d \rfloor) \cdot 2^{-t}}{4d} \\ &\leq \frac{f(k, t)}{4d}. \end{aligned}$$

Second, suppose $f(k - \delta, t) = h_1(k - \delta)$ for all $\delta \in \{1, \dots, d\}$. Then

$$\begin{aligned} \prod_{j=1}^d f(k - j, t) &= \prod_{j=1}^d \frac{\exp(-F_d(k - j - d + 1))}{64d} \\ &= \frac{\exp\left(-\sum_{j=1}^d F_d(k - j - d + 1)\right)}{(64d)^d} \\ &\leq \frac{\exp(-F_d(k - d + 1))}{(4d)(64d)} \\ &= \frac{h_1(k)}{4d} \\ &\leq \frac{f(k, t)}{4d}. \end{aligned}$$

The refinement from f to f' affects the above proof only if $f'(k, t) \neq f(k, t)$, since otherwise, $f(k - \delta, t) = f'(k - \delta, t)$ for all $0 \leq \delta \leq d$, so that the above arguments hold. If $f'(k, t) \neq f(k, t)$ then $f'(k, t) = n^{-0.9}/4$. In this case, either $f'(k - 1, t)$ might take the value $n^{-0.9}/4$ as well or it takes the value $f(k - 1, t)$. In the latter case, B3 follows by the same arguments as before, if we additionally apply $f(k, t) \leq n^{-0.9}/4 = f'(k, t)$. If both $f'(k, t)$ and $f'(k - 1, t)$ take the value $n^{-0.9}/4$, then

$$\prod_{j=1}^d f(k - j, t) \leq \frac{n^{-0.9}}{4} \cdot \prod_{j=2}^d f(k - j, t) \leq \frac{n^{-0.9}}{4} \cdot \frac{1}{4d} = \frac{f(k, t)}{4d}.$$

Thus, B1, B2, and B3 hold for f and f' . B4 does not hold for f . However, our refinement explicitly ensures this property for f' . \square

Based on these properties we prove now that the following invariants hold w.h.p. for every $t \in \{0, \dots, \log_2 n\}$. We say that a ball has *index k at time t* if the ball belongs to one of the batches $1, \dots, t$ and it is placed in group $k \bmod d$ with height $\lfloor (\Gamma + t + k)/d \rfloor$.

- $H_1(t)$: $\nu_{i,j}^{(t)} \leq f'(id + j, t) \cdot n/d$ for $i \geq 0, 0 \leq j < d$.
- $H_2(t)$: The number of balls with index at least $\ell_t + d$ at time t is bounded from above by a constant term c .

Observe that these invariants imply the bounds given in Theorem 1.5 as the function $f'(i, \log_2 n)$ decreases ‘‘Fibonacci exponentially’’ in i as discussed above.

We show the invariants H_1 and H_2 by an induction on the number of rounds t . Lemma 6.1 gives that the invariants hold at time 0. In the following, we prove that $H_1(t)$ and $H_2(t)$ hold w.h.p. assuming that $H_1(t - 1)$ and $H_2(t - 1)$ are given. Fix $t \in \{1, \dots, \log_2 n\}$. First, we consider $H_1(t)$. We prove this invariant by a further induction on $k = id + j$. Observe that we need only to prove the invariant for $k < \ell_t + d$ as the upper bound given for $k \geq \ell_t + d$ is a direct consequence of invariant H_2 . For $k \in \{0, \dots, d - 1\}$, property B1 gives $f'(k, t) = h_0(0)$. Hence, for $k < d$, invariant H_1 follows again directly from Lemma 6.1.

Now assume $d \leq k < \ell_t + d$. Suppose $H_1(t)$ is shown for all $k' < k$. For $i = \lfloor k/d \rfloor$ and $j = k \bmod d$, let $q(k) = q(i, j)$ denote the number of bins of group j containing $\Gamma + t + i$ balls already at the beginning of round t , and let $p(k) = p(i, j)$ denote the number of balls from batch t that are placed into a bin of group j that contains at least $\Gamma + t + i - 1$ balls. Clearly,

$$\nu_{i,j}^{(t)} \leq q(k) + p(k).$$

In the following, we calculate upper bounds for $q(k)$ and $p(k)$.

Observe that $q(k) = q(i, j)$ corresponds to $\nu_{i+1,j}^{(t-1)}$. Hence, invariant $H_1(t-1)$ gives

$$q(k) \leq f'((i+1)d + j, t-1) \cdot \frac{n}{d} \leq f'(k+d, t-1) \cdot \frac{n}{d} \stackrel{\text{(B2)}}{\leq} 0.5 \cdot f'(k, t) \cdot \frac{n}{d}$$

for $d \leq k < \ell_t + d$.

The term $p(k) = p(i, j)$ can be estimated as follows. If a ball is placed into a bin of group j with $\Gamma + t + i - 1$ balls, the d possible locations for that ball fulfill the following conditions. The randomly picked location from group g , $0 \leq g < j$, points to a bin with load at least $\Gamma + t + i$. (Otherwise, the always-go-left scheme would assign the ball to that location instead of location j .) At the ball's insertion time the number of these bins is at most $\nu_{i,g}^{(t)}$. By the induction on k , $\nu_{i,g}^{(t)} \leq f'(i \cdot d + g, t) \cdot n/d$. Thus, the probability that the location points to a suitable bin is at most $f'(i \cdot d + g, t)$. Furthermore, the randomly picked location from group g , $j \leq g < d$, points to a bin with load at least $\Gamma + t + i - 1$. At the ball's insertion time, the number of these bins is at most $\nu_{i-1,g}^{(t)}$. Thus, the probability for this event is at most $f'((i-1) \cdot d + g, t)$. Now multiplying the probabilities for all d locations yields that the probability that a fixed ball is allocated to group j with height $\Gamma + t + i$ or larger is at most

$$\prod_{g=0}^{j-1} f'(i \cdot d + g, t) \cdot \prod_{g=j}^{d-1} f'((i-1) \cdot d + g, t) = \prod_{g=1}^d f'(k-g, t) \stackrel{\text{(B3)}}{\leq} \frac{f'(k, t)}{4d}$$

for $d \leq k < \ell_t + d$. Taking into account all n balls of batch t , we obtain $\mathbf{E}[p(k)] \leq n \cdot f'(k, t)/(4d)$. Applying a Chernoff bound yields

$$\Pr \left[p(k) \geq 2n \cdot \frac{f'(k, t)}{4d} \right] \leq \exp \left(-n \cdot \frac{f'(k, t)}{8d} \right) \stackrel{\text{(B4)}}{\leq} \exp \left(-\frac{n^{0.1}}{32d} \right).$$

As a consequence, $p(k) \leq 0.5f'(k, t) \cdot n/d$, w.h.p.

Combining the bounds on $q(k)$ and $p(k)$ gives

$$\nu_{i,j}^{(t)} \leq q(k) + p(k) \leq f'(k, t) \cdot \frac{n}{d} \leq f'(id + j, t) \cdot \frac{n}{d}$$

for $d \leq k = id + j < \ell_t + d$. Thus, invariant $H_1(t)$ is shown.

Now we turn to the proof of $H_2(t)$. For $s \geq 0$, let L_s denote the number of balls with index at least $\ell_s + d$ at time s . Using this notation, invariant $H_2(t)$ states that $L_t \leq c$. For $s \geq r \geq 1$, let $L_s(r)$ denote the number of balls from batch r with index at least $\ell_s + d$ at time s . We claim

$$L_t = \sum_{s=1}^t L_t(s) \leq \sum_{s=1}^t L_s(s).$$

The first equation follows directly from the definition. The second equation can be seen as follows. First, observe that ℓ_{s-1} might be larger than ℓ_s as the function f' decreases over time. However, property B2 combined with the fact that f' decreases by at most a factor of two from time $s-1$ to time s yields $\ell_{s-1} \leq \ell_s + d$ for every $s \geq 0$, which implies $L_s(r) \leq L_{s-1}(r)$ for every $r \leq s-1$. By induction, we obtain $L_t(r) \leq L_s(r)$ for $r \leq s \leq t$ and especially $L_t(s) \leq L_s(s)$.

Let us study the probability that a fixed ball from batch s has index at least $\ell_s + d$ at time s and, hence, contributes to $L_s(s)$. This event happens only if each of the d randomly selected locations of the ball points to a bin whose topmost ball has index at least $\ell_s + d - d = \ell_s$. Invariant $H_1(s)$ yields that the fraction of such bins in each group is at most $f(\ell_s, s) \leq n^{-0.9}$. Thus, the probability that a ball from batch s contributes to $L(s)$ is at least $n^{-0.9d} \leq n^{-1.8}$. Now let us estimate the probability that there exist c balls from the batches 1 to t that fulfill this condition. This probability is at most

$$\binom{tn}{c} \cdot \left(\frac{1}{n^{1.8}}\right)^c \leq \left(\frac{\log_2 n}{cn^{0.8}}\right)^c \leq n^{-c/2},$$

where the last inequality holds for sufficiently large n . Consequently, with probability at least $1 - n^{-c/2}$, $L_t \leq \sum_{s=1}^t L_s(s) \leq c$. Thus, we have shown that $H_1(0), \dots, H_1(t)$ imply $H_2(t)$, w.h.p. This completes the proof for the case $m \geq n \log_2 n$.

Finally, let us investigate the case $m < n \log_2 n$. We break the set of balls into at most $t \leq \log n$ batches. All batches except for the last one contain exactly n balls; only the last batch might contain less. We use a simplified variant of the above analysis. In particular, we define $f(k, t) = h_1(k)$ instead of $f(k, t) = \max\{h_0(\lfloor k/d \rfloor) \cdot 2^{-t}, h_1(k)\}$. The invariants H_1 and H_2 can be shown by the same arguments as before. The advantage is that the identity between $f(k, t)$ and $h_1(k)$ is given from the beginning on, so that one does not have to iterate for $\log_2 n$ batches until the two functions become similar. In other words, the invariants H_1 and H_2 imply the bounds described in the theorem already after the first batch as well as after all subsequent batches. This completes the proof of Theorem 1.5.

7. Conclusions. We have presented the first tight analysis of two balls-into-bins multiple-choice processes: the greedy protocol of Azar et al. [1] and the always-go-left scheme due to Vöcking [33]. We showed that these schemes result in a maximum load (w.h.p.) of only $\frac{m}{n} + \frac{\ln \ln n}{\ln d} + \Theta(1)$ and $\frac{m}{n} + \frac{\ln \ln n}{d \ln \phi_d} + \Theta(1)$, respectively. Both these bounds are tight up to additive constants. In addition, we have given upper bounds on the number of bins with any given load. Furthermore, we presented the first comparative study of the two multiple-choice algorithms and gave a majorization result showing that the always-go-left scheme obtains a stochastically better load balancing than the greedy scheme for any choice of d , n , and m .

Our important technical contribution is the Short Memory Lemma, which informally states that the multiple-choice processes quickly “forget” their initial distribution of balls. The great consequence of this property is that the deviation of the multiple-choice processes from the optimal allocation (that is, the allocation in which each bin has either $\lfloor m/n \rfloor$ or $\lceil m/n \rceil$ balls) does not increase with the number of balls as in the case of the single-choice process. This property played a fundamental role in our analysis. We also hope that it will find further applications in the analysis of allocation processes. In particular, we believe that the use of the Markov chain approach to estimate the mixing time of underlying stochastic processes will be an important tool in analyzing other balls-into-bins or similar processes.

Our calculations in section 2 use some help from computers; it would be interesting to come up with a more elegant analysis that would possibly provide more insight on the greedy process for a polynomial number of balls.

Acknowledgments. We thank anonymous referees for helpful suggestions. We are also grateful to Christian Scheideler for inspiring discussion about the paper.

REFERENCES

- [1] Y. AZAR, A. Z. BRODER, A. R. KARLIN, AND E. UPFAL, *Balanced allocations*, SIAM J. Comput., 29 (1999), pp. 180–200.
- [2] M. ADLER, P. BERENBRINK, AND K. SCHRÖDER, *Analyzing an infinite parallel job allocation process*, in Proceedings of the 6th Annual European Symposium on Algorithms, Venice, Italy, G. Bilardi, G. F. Italiano, A. Pietracaprina, and G. Pucci, eds., Lecture Notes in Comput. Sci. 1461, Springer-Verlag, Berlin, 1998, pp. 417–428.
- [3] M. ADLER, S. CHAKRABARTI, M. MITZENMACHER, AND L. RASMUSSEN, *Parallel randomized load balancing*, Random Structures Algorithms, 13 (1998), pp. 159–188.
- [4] D. ALDOUS, *Random walks of finite groups and rapidly mixing Markov chains*, in Séminaire de Probabilités XVII, 1981/82, J. Azéma and M. Yor, eds., Lecture Notes in Math. 986, Springer-Verlag, Berlin, 1983, pp. 243–297.
- [5] P. BERENBRINK, A. CZUMAJ, T. FRIEDETZKY, AND N. D. VVEDENSKAYA, *On the analysis of infinite parallel job allocation processes via differential equations*, in Proceedings of the 11th Annual ACM Symposium on Parallel Algorithms and Architectures, Bar Harbor, ME, 2000, ACM Press, New York, 2000, pp. 99–108.
- [6] P. BERENBRINK, F. MEYER AUF DER HEIDE, AND K. SCHRÖDER, *Allocating weighted jobs in parallel*, Theory Comput. Syst., 32 (1999), pp. 361–386.
- [7] R. BUBLEY AND M. DYER, *Path coupling: A technique for proving rapid mixing in Markov chains*, in Proceedings of the 38th IEEE Symposium on Foundations of Computer Science, Miami Beach, FL, 1997, IEEE Computer Society Press, Los Alamitos, CA, 1997, pp. 223–231.
- [8] R. COLE, A. FRIEZE, B. M. MAGGS, M. MITZENMACHER, A. W. RICHA, R. K. SITARAMAN, AND E. UPFAL, *On balls and bins with deletions*, in Proceedings of the 2nd International Workshop on Randomization and Approximation Techniques in Computer Science, Barcelona, Spain, 1998, Lecture Notes in Comput. Sci. 1518, M. Luby, J. Rolim, and M. Serna, eds., Springer-Verlag, Berlin, 1998, pp. 145–158.
- [9] R. COLE, B. M. MAGGS, F. MEYER AUF DER HEIDE, M. MITZENMACHER, A. W. RICHA, K. SCHRÖDER, R. K. SITARAMAN, AND B. VÖCKING, *Randomized protocols for low-congestion circuit routing in multistage interconnection networks*, in Proceedings of the 30th Annual ACM Symposium on Theory of Computing, Dallas, TX, 1998, ACM Press, New York, 1999, pp. 378–388.
- [10] A. CZUMAJ, C. RILEY, AND C. SCHEIDELER, *Perfectly balanced allocation*, in Proceedings of the 7th International Workshop on Randomization and Approximation Techniques in Computer Science, Princeton, NJ, 2003, Lecture Notes in Comput. Sci. 2764, S. Arora, K. Jansen, J. D. P. Rolim, and A. Sahai, eds., Springer-Verlag, Berlin, 2003, pp. 240–251.
- [11] A. CZUMAJ AND V. STEMANN, *Randomized allocation processes*, Random Structures Algorithms, 18 (2001), pp. 297–331.
- [12] A. CZUMAJ, *Recovery time of dynamic allocation processes*, Theory Comput. Syst., 33 (2000), pp. 465–487.
- [13] A. CZUMAJ, P. KANAREK, M. KUTYŁOWSKI, AND K. LORYŚ, *Delayed path coupling and generating random permutations via distributed stochastic processes*, in Proceedings of the 10th Annual ACM–SIAM Symposium on Discrete Algorithms, Baltimore, MD, 1999, ACM, New York, SIAM, Philadelphia, PA, 1999, pp. 271–280.
- [14] D. DUBHASHI AND D. RANJAN, *Balls and bins: A study in negative dependence*, Random Structures Algorithms, 13 (1998), pp. 99–124.
- [15] N. L. JOHNSON AND S. KOTZ, *Urn Models and Their Application: An Approach to Modern Discrete Probability Theory*, John Wiley and Sons, New York, 1977.
- [16] R. M. KARP, M. LUBY, AND F. MEYER AUF DER HEIDE, *Efficient PRAM simulation on a distributed memory machine*, Algorithmica, 16 (1996), pp. 517–542.
- [17] V. F. KOLCHIN, B. A. SEVAST’YANOV, AND V. P. CHISTYAKOV, *Random Allocations*, V. H. Winston and Sons, Washington, D.C., 1978.
- [18] J. KORST, *Random duplicated assignment: An alternative to striping in video servers*, in Proceedings of the 5th ACM International Multimedia Conference, Seattle, WA, 1997, ACM Press, New York, 1997, pp. 219–226.

- [19] M. LUBY, D. RANDALL, AND A. SINCLAIR, *Markov chain algorithms for planar lattice structures*, in Proceedings of the 36th IEEE Symposium on Foundations of Computer Science, Milwaukee, WI, 1995, IEEE Computer Society Press, Los Alamitos, CA, 1995, pp. 500–509.
- [20] M. J. LUCZAK AND E. UPFAL, *Reducing network congestion and blocking probability through balanced allocation*, in Proceedings of the 40th IEEE Symposium on Foundations of Computer Science, New York, NY, 1999, IEEE Computer Society Press, Los Alamitos, CA, 1999, pp. 587–595.
- [21] C. MCDIARMID, *Concentration*, in Probabilistic Methods for Algorithmic Discrete Mathematics, M. Habib, C. McDiarmid, J. Ramirez-Alfonsin, and B. Reed, eds., Algorithms and Combinatorics, Springer-Verlag, Berlin, 1998, pp. 195–247.
- [22] M. MITZENMACHER, *Load balancing and density dependent jump Markov processes*, in Proceedings of the 37th IEEE Symposium on Foundations of Computer Science, Burlington, VT, 1996, IEEE Computer Society Press, Los Alamitos, CA, 1996, pp. 213–222.
- [23] M. MITZENMACHER, *The Power of Two Choices in Randomized Load Balancing*, Ph.D. thesis, Computer Science Department, University of California at Berkeley, Berkeley, CA, 1996.
- [24] M. MITZENMACHER, *Analyses of load stealing models based on differential equations*, in Proceedings of the 10th Annual ACM Symposium on Parallel Algorithms and Architectures, Puerto Vallarta, Mexico, 1998, ACM Press, New York, 1998, pp. 212–221.
- [25] M. MITZENMACHER, *Studying balanced allocations with differential equations*, *Combin. Probab. Comput.*, 8 (1999), pp. 473–482.
- [26] M. MITZENMACHER, *On the analysis of randomized load balancing schemes*, *Theory Comput. Syst.*, 32 (1999), pp. 292–301.
- [27] M. MITZENMACHER, B. PRABHAKAR, AND D. SHAH, *Load balancing with memory*, in Proceedings of the 43rd IEEE Symposium on Foundations of Computer Science, Vancouver, BC, Canada, 2002, IEEE Computer Society Press, Los Alamitos, CA, 2002, pp. 799–808.
- [28] M. MITZENMACHER, A. W. RICHA, AND R. SITARAMAN, *The power of two random choices: A survey of techniques and results*, in Handbook on Randomized Algorithms, Volume I, S. Rajasekaran, P. M. Pardalos, J. H. Reif, and J. Rolim, eds., Kluwer, Dordrecht, The Netherlands, 2001, pp. 255–294.
- [29] M. MITZENMACHER AND B. VÖCKING, *The asymptotics of selecting the shortest of two, improved*, in Proceedings of the 37th Allerton Conference on Communication, Control, and Computing, Urbana, IL, 1999, pp. 326–327.
- [30] M. RAAB AND A. STEGER, *“Balls into bins”—a simple and tight analysis*, in Proceedings of the 2nd International Workshop on Randomization and Approximation Techniques in Computer Science, Barcelona, Spain, 1998, M. Luby, J. Rolim, and M. Serna, eds., Lecture Notes in Comput. Sci. 1518, Springer-Verlag, Berlin, 1998, pp. 159–170.
- [31] V. STEMANN, *Parallel balanced allocations*, in Proceedings of the 8th Annual ACM Symposium on Parallel Algorithms and Architectures, Padua, Italy, 1996, ACM Press, New York, 1996, pp. 261–269.
- [32] P. SANDERS, S. EGNER, AND J. KORST, *Fast concurrent access to parallel disks*, *Algorithmica*, 35 (2003), pp. 21–55.
- [33] B. VÖCKING, *How asymmetry helps load balancing*, *J. ACM*, 50 (2003), pp. 568–589.
- [34] N. D. VVEDENSKAYA, R. L. DOBRUSHIN, AND F. I. KARPELEVICH, *Queueing system with selection of the shortest of two queues: An asymptotic approach*, *Probl. Inform. Transm.*, 32 (1996), pp. 15–27.
- [35] N. D. VVEDENSKAYA AND Y. M. SUHOV, *Dobrushin’s Mean-Field Approximation for Queue with Dynamic Routing*, Technical report 3328, INRIA, Le Chesnay, France, 1997.

LINEARIZATION AND COMPLETENESS RESULTS FOR TERMINATING TRANSITIVE CLOSURE QUERIES ON SPATIAL DATABASES*

FLORIS GEERTS[†], BART KUIJPERS[†], AND JAN VAN DEN BUSSCHE[†]

Abstract. We study queries to spatial databases, where spatial data are modeled as semi-algebraic sets, using the relational calculus with polynomial inequalities as a basic query language. We work with the extension of the relational calculus with terminating transitive closures. The main result is that this language can express the linearization of semialgebraic databases. We also show that the sublanguage with linear inequalities only can express all computable queries on semilinear databases. As a consequence of these results, we obtain a completeness result for topological queries on semialgebraic databases.

Key words. constraint databases, real algebraic geometry, transitive closure logics, query languages

AMS subject classifications. 68P15, 14P10, 57R05, 03B70

DOI. 10.1137/S0097539702410065

1. Introduction. Spatial database systems [1, 8, 12, 24, 25, 42] are concerned with the representation and manipulation of data that have a geometric or topological interpretation. Conceptually, spatial databases store geometric figures, which are possibly infinite sets of points in a real space \mathbf{R}^n . The framework of constraint databases [34], introduced by Kanellakis, Kuper, and Revesz [27], provides an elegant and powerful model for spatial databases. In the setting of the constraint model, a geometric figure is finitely represented as a Boolean combination of polynomial equalities and inequalities over the real numbers. Such figures are known as semi-algebraic sets. Special cases of figures definable by linear polynomials are known as semilinear sets [6].

The relational calculus or first-order logic, expanded with polynomial equalities and inequalities and evaluated over the semialgebraic sets (viewed as relations over the reals) stored in the database, serves as a basic spatial query language and is denoted by FO+POLY. The special case of queries expressed using linear equalities and inequalities is denoted by FO+LIN. Several authors have argued that the restriction to linear polynomial constraints provides a sufficiently general framework for spatial database applications [21, 46, 47]. Indeed, in geographic information systems (GIS), which form one of the main application areas of spatial databases, linear representations are used to model spatial objects [34, Chapter 9]. Existing implementations of the constraint model, for instance, the work on the system DEDALE [19, 20, 21], are also restricted to linear polynomial constraints. Indeed, for these constraints, the evaluation of queries expressed in FO+LIN is conceptually easier and can be computed by numerous efficient algorithms for geometric operations on linear figures [38]. The computational complexity of evaluating an FO+LIN query on linear constraint

*Received by the editors June 22, 2002; accepted for publication (in revised form) November 13, 2005; published electronically April 7, 2006.

<http://www.siam.org/journals/sicomp/35-6/41006.html>

[†]Theoretical Computer Science Group, Hasselt University and Transnational University of Limburg, Agoralaan, Gebouw D, B-3590 Diepenbeek, Belgium (floris.geerts@uhasselt.be, bart.kuijpers@uhasselt.be, jan.vandenbussche@uhasselt.be). The first author is a postdoctoral researcher of the FWO-Vlaanderen.

databases (NC^1) is also slightly lower than that of evaluating an $\text{FO}+\text{POLY}$ query on polynomial constraint databases (NC) [2, 22, 41].

Since the expressive power of the basic query languages $\text{FO}+\text{POLY}$ and $\text{FO}+\text{LIN}$ is rather limited [34, Chapters 5 and 6], it makes sense to consider more powerful extensions.

Various extensions with recursion have already been introduced and studied. Grumbach and Kuper [18] defined syntactic variants of DATALOG with linear constraints which capture exactly the queries on linear constraint databases in the plane, which have PTIME and PSPACE data complexity. Kreutzer [30] defines several recursive languages capturing PTIME and PSPACE on a restricted class of linear constraint databases. Termination properties of DATALOG with polynomial constraints are investigated by Kuijpers et al. [31] and Kuijpers and Smits [33].

In this paper, we study the expressive power of $\text{FO}+\text{POLY}$ (and $\text{FO}+\text{LIN}$) extended with the transitive closure operator (TC). Transitive closure is a simple form of recursion and we apply it only in a simple way; specifically, we do not apply TC to formulas with extra free variables (parameters), as is allowed in the standard definition of transitive closure logic [11].

In the first part of the paper, we show that when we extend the TC operator with explicit stop conditions, which we denote by TCS , the language $\text{FO}+\text{LIN}+\text{TCS}$ is computationally complete on the class of databases definable by linear polynomials with integer coefficients (\mathbf{Z} -linear databases). This means that for every partial computable query Q , there is a formula φ such that for every \mathbf{Z} -linear database D , the evaluation of φ on D terminates if and only if $Q(D)$ is defined and results in $Q(D)$. It remains an open problem whether $\text{FO}+\text{LIN}+\text{TC}$ (without explicit stop conditions) is also computationally complete in this sense. We point out that recently, Kreutzer [29] defined an extension of $\text{FO}+\text{LIN}$ with a different transitive closure operator and proved completeness on linear constraint databases as well (see the end of section 3 for more details).

In the second part of the paper, we investigate the expressive power of $\text{FO}+\text{POLY}+\text{TCS}$ on general polynomial constraint databases. In contrast to the linear case, we have not been able to establish the computational completeness. Yet, we will show that the language is complete as far as all Boolean topological queries are concerned.

In order to prove this result, we show that there is a formula of $\text{FO}+\text{POLY}+\text{TC}$ (no stop conditions are needed) that expresses *linearization*: when evaluated on an arbitrary semialgebraic set A , it results in a semilinear set \hat{A} topologically equivalent (i.e., homeomorphic) to A . Moreover, \hat{A} can be assumed to be a \mathbf{Z} -linear set.

Our linearization formula always terminates, in the sense that on any input A , every application of the TC operator in the formula converges after a finite number of stages. In the case when A is bounded, the linearization formula can be sharpened to produce a set \hat{A} that is arbitrarily close to the input set A .

The components of the linearization formula require a number of new geometric constructions in $\text{FO}+\text{POLY}$. More specifically, we introduce the uniform cone radius decomposition of semialgebraic sets. Using the result of Geerts [14], we show that this decomposition can be defined in $\text{FO}+\text{POLY}$. Also, we define the regular decomposition of semialgebraic sets and use the results of Rannou [39] to show that this decomposition is expressible in $\text{FO}+\text{POLY}$.

The linearization algorithm also implies that semialgebraic sets in \mathbf{R}^n can be linearized, a fact which has been known for a long time [7]. The standard constructive linearization (or triangulation) algorithm for semialgebraic sets, which is attributed

to Hardt [26], can be found in the standard textbook on real algebraic geometry [6, section 9.2] and in the more recent book on algorithms in real algebraic geometry [3, Chapter 5].

The difference of the existing linearization algorithm for semialgebraic sets is that the polynomials appearing in the description of the semialgebraic sets are used explicitly. This is not possible in our setting because we can only interact with the semialgebraic set using queries. Because of this, our algorithm is not likely to be as efficient as the existing algorithm (we did not compute the exact complexity though). Moreover, our linearization is based on the local conical behavior of semialgebraic sets, and the inductive construction based on these cones might be of interest in real algebraic geometry.

Finally, we use the linearization formula in the following two ways to show the expressibility in FO+POLY+TC of two common queries which are known to be not expressible in FO+POLY: (1) We show that the connectivity query on polynomial constraint databases is expressible by an always terminating formula in FO+POLY+TC; (2) we show that there is a formula in FO+POLY+TC that always has a terminating evaluation and that evaluates on a given bounded semialgebraic set A to a number that is arbitrarily close to the volume of A .

We remark that some of the above results were already described (in considerably less detail) for two dimensions [16] and arbitrary dimensions [13].

This paper is organized as follows. Section 2 gives the definition of polynomial constraint databases and defines the standard first-order query languages. Section 3 extends these languages with a transitive closure operator. Section 4 studies the computational completeness of these extensions and gives some inexpressibility results of the first-order query languages. Section 5 provides geometric tools necessary for the linearization construction. Section 6 presents the construction itself and discusses applications of linearization (testing connectivity and approximating the volume).

2. Preliminaries. We denote the set of real numbers by \mathbf{R} , the set of algebraic numbers by \mathbf{A} , the set of integers by \mathbf{Z} , and the set of natural numbers by \mathbf{N} .

A *semialgebraic set in \mathbf{R}^n* is a finite union of sets definable by conditions of the form

$$f_1(\vec{x}) = f_2(\vec{x}) = \dots = f_k(\vec{x}) = 0, \quad g_1(\vec{x}) > 0, \quad g_2(\vec{x}) > 0, \dots, g_\ell(\vec{x}) > 0,$$

where $\vec{x} = (x_1, \dots, x_n) \in \mathbf{R}^n$, and where $f_1(\vec{x}), \dots, f_k(\vec{x}), g_1(\vec{x}), \dots, g_\ell(\vec{x})$ are multivariate polynomials in the variables x_1, \dots, x_n with integer coefficients. A *\mathbf{Z} -linear (A-linear) set in \mathbf{R}^n* is a semialgebraic set which can be defined in terms of linear polynomials with *integer (algebraic)* coefficients.

A *database schema \mathcal{S}* is a finite set of relation names, each with a given arity. A *polynomial constraint database D over \mathcal{S}* assigns to each $S \in \mathcal{S}$ a semialgebraic set S^D in \mathbf{R}^k , where k is the arity of S . A *\mathbf{Z} -linear (A-linear) constraint database* assigns to each $S \in \mathcal{S}$ a \mathbf{Z} -linear (A-linear) set S^D in \mathbf{R}^k , where k is the arity of S . A *k -ary query over \mathcal{S}* is a partial function Q that maps each database D over \mathcal{S} to a k -ary relation $Q(D) \subseteq \mathbf{R}^k$.

First-order logic over the vocabulary $(+, \times, 0, 1, <)$ expanded with the database schema \mathcal{S} provides a basic query language which we denote by FO+POLY. The sublanguage of FO+POLY consisting of the formulas that do not use multiplication is denoted by FO+LIN.

Every formula $\varphi(x_1, \dots, x_k)$ in FO+POLY expresses a k -ary query as follows: Let D be a database over \mathcal{S} ; then

$$\varphi(D) = \{(a_1, \dots, a_k) \in \mathbf{R}^k \mid \langle \mathbf{R}, D \rangle \models \varphi(a_1, \dots, a_k)\}.$$

Here, by $\langle \mathbf{R}, D \rangle$ we mean the standard structure of the reals $\langle \mathbf{R}; +, \times, 0, 1, < \rangle$ expanded with the relations (semialgebraic sets) in D .

Example 2.1. Suppose that \mathcal{S} contains the binary relation name S . Then the FO+POLY formula

$$\varphi(x, y) \equiv \exists \varepsilon \forall x' \forall y' (\varepsilon > 0 \wedge ((x - x')^2 + (y - y')^2 < \varepsilon \rightarrow S(x', y')))$$

expresses the query that maps any database D over \mathcal{S} to the interior of S^D . \diamond

FO+POLY queries can be effectively evaluated as follows. Let $\varphi(x_1, \dots, x_k)$ be an FO+POLY formula over schema \mathcal{S} , and let D be a database over \mathcal{S} . For every $S \in \mathcal{S}$, we represent the set S^D by some quantifier-free polynomial constraint formula $\psi_S(y_1, \dots, y_k)$, where k is the arity of S , that defines S^D in the sense that $S^D = \{(a_1, \dots, a_k) \in \mathbf{R}^k \mid \mathbf{R} \models \psi_S(a_1, \dots, a_k)\}$. Now replace in φ every subformula of the form $S(z_1, \dots, z_k)$ with $\psi_S(z_1, \dots, z_k)$. Making these replacements for every $S \in \mathcal{S}$, we obtain a polynomial constraint formula which we denote by φ^D and which defines $\varphi(D)$ in the sense that $\varphi(D) = \{(a_1, \dots, a_k) \in \mathbf{R}^k \mid \mathbf{R} \models \varphi^D(a_1, \dots, a_k)\}$.

Because first-order logic over the reals admits quantifier elimination [43], we can rewrite φ^D in a quantifier-free form from which we can conclude that $\varphi(D)$ is always a semialgebraic set. This is called the closure principle. The reals without multiplication also admit quantifier elimination, so in the same way, if D is semilinear and φ is in FO+LIN, then $\varphi(D)$ is also semilinear. Thus, there is also a closure principle for FO+LIN provided we work with semilinear databases. For more information on FO+POLY and FO+LIN queries, we refer the reader to the literature [34].

3. Transitive closure logics. Many interesting spatial database queries are not expressible in the first-order query languages FO+POLY and FO+LIN, e.g., the query that asks whether a given set is topologically connected or not. Therefore, it makes sense to consider extensions of FO+POLY (or FO+LIN) with recursion to obtain more powerful query languages. We study one of the most simple recursion constructs in this context, i.e., the transitive closure operator TC.

An immediate observation is that TC cannot be added “just like that” with its standard mathematical semantics without losing the important closure principle.

Example 3.1. The transitive closure of the semialgebraic set $\{(x, y) \in \mathbf{R}^2 \mid y = 2x\}$ equals $\{(x, y) \in \mathbf{R}^2 \mid \exists i \in \mathbf{N} : y = 2^i x\}$, which is not a semialgebraic set. \diamond

Therefore, we look at the TC operator quite naturally as a programming construct with a purely operational semantics. For example, we will look at the transitive closure example just mentioned simply as a nonterminating computation. Almost all programming languages allow for the expression of nonterminating computations, and it is part of the programmer’s job to avoid writing such programs.

A formula in FO+POLY+TC is a formula built in the same way as an FO+POLY formula, but with the following extra formation rule: If $\psi(\vec{x}, \vec{y})$ is a formula with \vec{x}, \vec{y} k -tuples of variables, and \vec{s}, \vec{t} are k -tuples of terms, then

$$(3.1) \quad [\text{TC}_{\vec{x}; \vec{y}} \psi](\vec{s}, \vec{t})$$

is also a formula which has as free variables those in \vec{s} and \vec{t} . Since the only free variables in $\psi(\vec{x}, \vec{y})$ are those in \vec{x} and \vec{y} , we do not allow parameters in applications

of the TC operator, as are allowed in general transitive closure logic studied in finite model theory [11]. With parameters, it is not so clear how to preserve the simple and elegant operational semantics we define next.

The semantics of a subformula of the above form (3.1) evaluated on a database D is defined in the following operational manner:

1. Evaluate, recursively, $\psi(D)$.
2. Start computing the following iterative sequence of $2k$ -ary relations:

$$\begin{aligned} X_0 &:= \psi(D), \\ X_{i+1} &:= X_i \cup \{(\vec{x}, \vec{y}) \in \mathbf{R}^{2k} \mid \exists \vec{z}(X_i(\vec{x}, \vec{z}) \wedge X_0(\vec{z}, \vec{y}))\}. \end{aligned}$$

Stop as soon as an i has been found such that $X_i = X_{i+1}$.

3. The semantics of $[\text{TC}_{\vec{x};\vec{y}} \psi](\vec{s}, \vec{t})$ is now defined as the $2k$ -ary relation X_i .

Since every step in the above algorithm, including the test for $X_i = X_{i+1}$, is expressible in FO+POLY, every step is effective and the only reason why the evaluation may not be effective is that the computation does not terminate. In that case the semantics of the formula (3.1) (and any other formula in which it occurs as subformula) is undefined.

The language FO+LIN+TC consists of all FO+POLY+TC formulas that do not use multiplication.

Example 3.2. Let S be a relation name of arity n . Consider the following FO+POLY+TC formula:

$$\mathbf{connected} \equiv \forall \vec{s} \forall \vec{t} ((S(\vec{s}) \wedge S(\vec{t})) \rightarrow [\text{TC}_{\vec{x};\vec{y}} \mathbf{lineconn}](\vec{s}, \vec{t})),$$

where $\mathbf{lineconn}(\vec{x}, \vec{y})$ is the formula

$$\forall \lambda (0 \leq \lambda \leq 1 \wedge \forall \vec{t} (\vec{t} = \lambda \vec{x} + (1 - \lambda) \vec{y} \rightarrow S(\vec{t}))).$$

In section 6.5, we will prove that the TC-subformula in $\mathbf{connected}$ terminates on all linear constraint databases over \mathcal{S} . Note that a pair of points (\vec{p}, \vec{q}) belongs to the TC of $\mathbf{lineconn}(D)$ (with D semilinear) if and only if \vec{p} and \vec{q} belong to the same connected component of S^D . Hence, $\mathbf{connected}$ effectively expresses connectivity of semilinear sets. \diamond

We will sometimes want to be able to specify an explicit termination condition on transitive closure computations. To this end, we introduce the language FO+POLY+TCS.

Formulas in FO+POLY+TCS are again built in the same way as in FO+POLY but with the following extra formation rule: If $\psi(\vec{x}, \vec{y})$ is a formula with \vec{x}, \vec{y} k -tuples of variables; σ is an FO+POLY sentence (formula without free variables) over the schema \mathcal{S} expanded with a special $2k$ -ary relation name X ; and \vec{s}, \vec{t} are k -tuples of terms, then

$$(3.2) \quad [\text{TC}_{\vec{x};\vec{y}} \psi \mid \sigma](\vec{s}, \vec{t})$$

is also a formula which has as free variables those in \vec{s} and \vec{t} . We call σ the *stop condition* of this formula.

The semantics of a subformula of the above form (3.2) evaluated on databases D is defined in the same manner as in the case without stop condition, but now we stop not only in case an i is found such that $X_i = X_{i+1}$, but also in case an i is found such that $(D, X_i) \models \sigma$, whichever case occurs first.

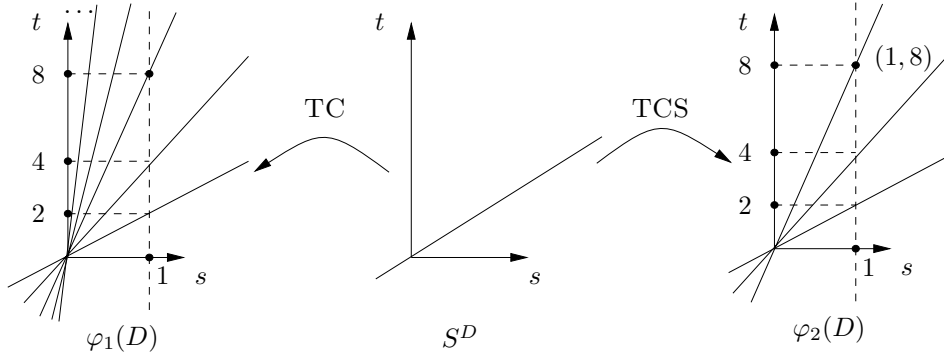


FIG. 3.1. Illustration of the difference between transitive closure without stop condition (left) and with stop condition (right).

Example 3.3. Let S be a relation name of arity n in \mathcal{S} , and consider the FO+POLY+TCS formula

$$(3.3) \quad \varphi_1(s, t) \equiv [\text{TC}_{x;y} S](s, t)$$

and the formula

$$(3.4) \quad \varphi_2(s, t) \equiv [\text{TC}_{x;y} S \mid X(1, 8)](s, t).$$

On the database D over S , where $S^D = \{(x, y) \in \mathbf{R}^2 \mid y = 2x\}$, the evaluation of formula (3.3) does not terminate, but formula (3.4) evaluates in three iterations to $\{(s, t) \in \mathbf{R}^2 \mid t = 2s \vee t = 4s \vee t = 6s \vee t = 8s\}$. An illustration is given in Figure 3.1. \diamond

The language FO+LIN+TCS consists of all FO+POLY+TCS formulas that do not use multiplication.

An alternative way of controlling the computation of the transitive closure is provided by Kreutzer [29]. He allows a parametrized transitive closure operator in which the computation of the transitive closure can be restricted to certain paths (after specifying certain starting points).

It can be easily seen that any formula in FO+LIN+TC or FO+POLY+TC can be expressed by an equivalent formula in the corresponding logics of Kreutzer (see Geerts and Kuijpers [17]). Moreover, the transitive closure logic FO+LIN+KTC (the “K” stands for “Kreutzer”) is computationally complete on \mathbf{Z} -linear constraint databases [29]. As we will see in the next section, the same completeness result holds for FO+LIN+TCS. Hence, FO+LIN+KTC and FO+LIN+TCS are equally expressive on \mathbf{Z} -linear constraint databases. Despite this similarity, the way in which queries are expressed in each language is quite different. Indeed, FO+LIN+KTC has an “a priori” character because *starting* points have to be properly selected in order to obtain terminating formula. In FO+LIN+TCS, termination is forced by the *stop* conditions, which are of an “a posteriori” character.

We point out that termination properties of these logics on general polynomial constraint databases have already been studied [17]. However, a complete comparison of these logics on polynomial constraint databases is left open.

4. Expressivity results. In this section, we show a general result on the expressive power of FO+LIN+TCS. More specifically, we prove that FO+LIN+TCS is computationally complete on \mathbf{Z} -linear constraint databases (Theorem 4.4). The proof consists of three steps. In the first step, we show that any computable function on

the natural numbers can be simulated in FO+LIN+TCS (Lemma 4.1). In the second step, we show that there exists an encoding of \mathbf{Z} -linear constraint databases by finite sets of rational numbers, and show that both the encoding and the corresponding decoding are expressible in FO+LIN+TCS (Lemmas 4.2 and 4.3). This implies that FO+LIN+TCS is computationally complete on \mathbf{Z} -linear constraint databases.

For polynomial constraint databases, we show that FO+POLY+TCS is computationally complete for Boolean topological queries. This follows from the completeness on \mathbf{Z} -linear constraint databases and the existence of an FO+POLY+TC query that, given any polynomial constraint database as input, returns a \mathbf{Z} -linear constraint database which is topologically equivalent to the input. In this section we show that this “linearization query” is not expressible in FO+POLY. The FO+POLY+TC construction of the linearization query will be presented in section 6 (following preparations in section 5.1).

4.1. Recursive functions on the natural numbers. We first show that FO+LIN+TCS is computationally complete on the set of natural numbers \mathbf{N} .

LEMMA 4.1. *For every partial computable function $f : \mathbf{N}^k \rightarrow \mathbf{N}$, there exists a formula $\varphi_f(y)$ in FO+LIN+TCS over the schema $\mathcal{S} = \{S\}$, with S a k -ary relation, such that for any database D over \mathcal{S} with $S^D = \{(n_1, \dots, n_k)\}$, we have that $\varphi_f(D)$ is defined if and only if $f(n_1, \dots, n_k)$ is defined, and in this case $\varphi_f(D) = \{f(n_1, \dots, n_k)\}$.*

Proof. We show this by simulating the run of a nondeterministic p -counter machine M_f which computes f . Here $M_f = (Q, \delta, q_0, q_f)$, where Q is a finite set of internal states, $q_0 \in Q$ is the initial state, and $q_f \in Q$ is the final (halting) state. The set δ contains quadruples of the form $[q, i, s, q'] \in Q \times \{1, \dots, p\} \times \{Z, P\} \times Q$ or $[q, i, d, q'] \in Q \times \{1, \dots, p\} \times \{-, +\} \times Q$. The quadruple $[q, i, s, q']$ means that if M_f is in state q and the i th counter is equal to zero (when $s = Z$) or positive (when $s = P$), then change the state into q' . The quadruple $[q, i, d, q']$ means that if M_f is in state q , then increase the i th counter by one (when $d = +$), or decrease the i th counter by one (when $d = -$), and change the state into q' . We assume that $Q = \{0, 1, \dots, m-1, m\}$, $q_0 = 0$, and $q_f = m$. Moreover, we assume that $p \geq k$ and that the initial configuration of M_f when computing $f(n_1, \dots, n_k)$ has n_1, \dots, n_k as the values of the first k counters. When a halting state is reached, we assume that the first counter contains $f(n_1, \dots, n_k)$.

We define the first-order formula $\Psi_{\text{step}}(q, n_1, \dots, n_p, q', n'_1, \dots, n'_p)$, which describes a single step in a run of M_f . The formula Ψ_{step} is the disjunction of the following formulas for $[q, i, s, q']$ and $[q, i, d, q']$ in δ :

$$\Psi_{[q,i,Z,q']} \equiv Q(q) \wedge Q(q') \wedge n'_i = n_i = 0 \wedge \bigwedge_{j \in \{1, \dots, i-1, i+1, \dots, p\}} n_j = n'_j,$$

$$\Psi_{[q,i,P,q']} \equiv Q(q) \wedge Q(q') \wedge n'_i = n_i > 0 \wedge \bigwedge_{j \in \{1, \dots, i-1, i+1, \dots, p\}} n_j = n'_j,$$

$$\Psi_{[q,i,+,q']} \equiv Q(q) \wedge Q(q') \wedge n'_i = n_i + 1 \wedge \bigwedge_{j \in \{1, \dots, i-1, i+1, \dots, p\}} n_j = n'_j,$$

$$\Psi_{[q,i,-,q']} \equiv Q(q) \wedge Q(q') \wedge n'_i = n_i - 1 \wedge \bigwedge_{j \in \{1, \dots, i-1, i+1, \dots, p\}} n_j = n'_j.$$

We use the stop condition σ , which checks whether the final state has been reached starting from the initial state:

$$\sigma \equiv \exists y_1 \cdots \exists y_p \exists n_1 \cdots \exists n_k (S(n_1, \dots, n_k) \wedge X(0, n_1, \dots, n_k, \vec{0}_{p-k}, m, y_1, \dots, y_p)).$$

Here, $\vec{0}_\ell$ denotes the ℓ -tuple $(0, \dots, 0)$.

The desired formula $\varphi_f(y)$ extracts $f(n_1, \dots, n_k)$ from the first counter (represented by the variable y) when the stop condition is satisfied:

$$\begin{aligned} &\exists y_2 \cdots \exists y_p \exists n_1 \cdots \exists n_k (S(n_1, \dots, n_k) \\ &\quad \wedge [\text{TC}_{q, \vec{n}; q', \vec{n}'} \Psi_{\text{step}} \mid \sigma](0, n_1, \dots, n_k, \vec{0}_{p-k}, m, y, y_2, \dots, y_p)). \quad \square \end{aligned}$$

4.2. Finite representation of \mathbf{Z} -linear constraint databases.

LEMMA 4.2. *There exists an encoding of \mathbf{Z} -linear constraint databases into finite relational databases over the rationals, and a corresponding decoding, which are both expressible in FO+LIN+TCS.*

Proof. It was shown by Vandeurzen [46] and Vandeurzen, Gyssens, and Van Gucht [48] that any \mathbf{Z} -linear set in \mathbf{R}^n has a finite geometric representation by means of a finite set over \mathbf{Q} consisting of $(n + 1)^2$ -ary tuples. Basically, this geometric representation contains the projective coordinates¹ of a complete triangulation of the \mathbf{Z} -linear set. Moreover, this representation can be expressed in FO+POLY. Vandeurzen [46] and Vandeurzen, Gyssens, and Van Gucht [48] actually show that this representation can be expressed in an extension of FO+LIN with some limited amount of multiplicative power. Also, the corresponding decoding, which computes the \mathbf{Z} -linear constraint database given its finite geometric representation, can be expressed in this logic.

Hence, the lemma follows if we can show that FO+LIN+TCS can perform this limited amount of multiplication.

More specifically, we have to be able to express the multiplication of rationals q_i from a finite set $S = \{q_1, \dots, q_m\}$ with a real number x , i.e., $q_i x$ for $i = 1, \dots, m$. First, we express how integers n_i and d_i can be computed in FO+LIN+TCS such that $q_i = \frac{n_i}{d_i}$ for $i = 1, \dots, m$.

We assume that all rational numbers in the set S are positive. The case of all negative rational numbers is completely analogous. If both positive and negative rational numbers occur in the set, we separate the positive from the negative and treat both sets separately.

Consider the following enumeration $enum$ of pairs of natural numbers: $enum$ is a mapping from $\mathbf{N} \times \mathbf{N}$ to $\mathbf{N} \times \mathbf{N}$ defined by

$$enum : (i, j) \mapsto \begin{cases} (i + 1, j - 1) & \text{if } j > 0, \\ (0, i + 1) & \text{if } j = 0. \end{cases}$$

For every pair $(p, q) \in \mathbf{N} \times \mathbf{N}$, there clearly exists $k \in \mathbf{N}$ such that $enum^k(0, 0) = (p, q)$. We shall interpret (p, q) as the rational number $\frac{p}{q}$ in case $q \neq 0$ and as 0 otherwise.

Given a rational number q and two natural numbers n and d , we can test in FO+LIN+TCS whether $q = \frac{n}{d}$. This test can be performed as follows. Let $frac : \mathbf{R}^3 \rightarrow \mathbf{R}^3$ be the mapping defined as

$$frac : (q, j, v) \mapsto (q, j - 1, v + q).$$

¹Projective coordinates are used to deal with unbounded databases and the unbounded simplices in their triangulation.

Then for given $q \in \mathbf{Q}$ and $n, d \in \mathbf{N}$, we have that $q = \frac{n}{d}$ if and only if $\text{frac}^d(q, d, 0) = (q, 0, n)$.

To find the numerator and denominator of a rational number q , we will enumerate all pairs of natural numbers $(n, d) = \text{enum}^k(0, 0)$, $k = 0, 1, \dots$ and test for each pair whether $\text{frac}^d(q, d, 0) = (q, 0, n)$. For this, we combine *enum* and *frac* into a partial mapping $\text{tryall} : \mathbf{R}^5 \rightarrow \mathbf{R}^5$ defined as

$$(q, i, j, u, v) \mapsto \begin{cases} (q, i, j, u', v') & \text{with } (q, u', v') = \text{frac}(q, u, v) \quad \text{if } u \geq 1, \\ (q, i', j', j', 0) & \text{with } (i', j') = \text{enum}(i, j) \quad \text{if } u = 0. \end{cases}$$

We claim that $q = \frac{n}{d}$ for $n, d \in \mathbf{N}$ if and only if $\text{tryall}^k(q, 0, 0, 0, 0) = (q, n, d, 0, n)$. Indeed, starting from $(q, 0, 0, 0, 0)$ the iterates of *tryall* behave as follows. Suppose we are at the k th iterate. If the third coordinate of $\text{tryall}^k(q, 0, 0, 0, 0)$ is zero, a new pair of natural numbers is generated (using the *enum* mapping). Assume that $\text{tryall}^{k+1}(q, 0, 0, 0, 0) = (q, i, j, j, 0)$ and suppose that $j > 0$ (otherwise we jump to a new pair of natural numbers immediately). Then, using the *frac* mapping, we end up after j more iterations at $\text{tryall}^{k+j+1}(q, 0, 0, 0, 0) = \text{tryall}^j(q, i, j, j, 0) = (q, i, j, 0, jq)$ (*frac* reduces the fourth coordinate by one in each iteration). Note that if $i = jq$, then we have found a numerator i and denominator j of q . In any case, we move on to $\text{tryall}^{k+j+2}(q, 0, 0, 0, 0) = (q, i', j', j', 0)$, where (i', j') is the next pair of natural numbers, and the above process starts again. In this way, the iterates of *tryall* visit every pair of natural numbers starting from $(q, 0, 0, 0, 0)$; between two consecutive pairs, it is checked whether the first pair is a numerator/denominator pair for q . The mapping *tryall* can clearly be expressed by an FO+LIN formula,

$$\psi_{\text{tryall}}(q, i, j, u, v, q', i', j', u', v'),$$

expressing that $\text{tryall}(q, i, j, u, v) = (q', i', j', u', v')$.

Let $\Psi(q, i, j, u, v, q', i', j', u', v')$ be the formula

$$q \geq 0 \wedge i \geq 0 \wedge j \geq 0 \wedge i' \geq 0 \wedge j' \geq 0 \wedge u \geq 0 \wedge q = q' \wedge \psi_{\text{tryall}}(q, i, j, u, v, q', i', j', u', v').$$

Given a finite set of rational numbers $S = \{q_1, \dots, q_m\}$, we obtain a denominator and numerator for all these numbers by taking the transitive closure

$$(4.1) \quad [\text{TC}_{q,i,j,u,v;q',i',j',u',v'} \Psi \mid \sigma](\vec{s}, \vec{t}),$$

where \vec{s} and \vec{t} are 5-tuples of variables, and where

$$\sigma \equiv \forall q(S(q) \rightarrow \exists n \exists d X(q, 0, 0, 0, 0, q, n, d, 0, n)).$$

This condition stops the computation of the transitive closure of Ψ when, for each rational number q in S , there exists a k such that $\text{tryall}^k(q, 0, 0, 0, 0) = (q, n, d, 0, n)$, or in other words, when a pair of natural numbers (n, d) has been encountered such that $q = \frac{n}{d}$. If multiple pairs (n, d) represent the same rational number in S , we select the pair with the smallest value of n . Thus, we obtain for each $q \in S$ a unique denominator and numerator.

We are now ready to show how to express the multiplication of rational numbers from a finite set S with a real number. By what we just showed, we may assume that

the rational numbers are represented as numerator/denominator pairs; i.e., we may assume that $S = \{(n_1, d_1), \dots, (n_m, d_m)\}$.

Let \max be the largest natural number occurring in S . We first compute any multiplication of the form rn with $r \in \mathbf{R}$ and $n \in \{0, 1, \dots, \max\}$.

For this, we define the following formula $\mathbf{natmult}(x, y, z, x', y', z')$:

$$\begin{aligned} x &= x' \wedge y' = y - 1 \wedge z' \\ &= z + x \wedge \exists \max(\exists n(S(\max, n) \vee S(n, \max)) \\ &\quad \wedge \forall n \forall d(S(n, d) \rightarrow n \leq \max \wedge d \leq \max) \wedge 0 \leq y \wedge y \leq \max). \end{aligned}$$

Then the formula

$$\mathbf{mult}(a, b, c) \equiv [\mathbf{TC}_{x,y,z;x',y',z'} \mathbf{natmult}](a, b, 0, a, 0, c)$$

holds if and only if $ab = c$ for $a \in \mathbf{R}$, $b \in \mathbf{N}$ and $b \leq \max$. In this way, we can retrieve any multiple up to \max of any real number.

Finally, we define $\mathbf{ratmult}(z, y, n, d) \equiv \exists u(\mathbf{mult}(z, d, u) \wedge \mathbf{mult}(y, n, u))$. This formula holds for (z, y, n, d) if and only if $z = yq$ with $z, y \in \mathbf{R}$, and $q = \frac{n}{d}$ with $(n, d) \in S$. \square

4.3. Natural number representation.

LEMMA 4.3. *There exists an encoding of finite relations over the rational numbers into single natural numbers, and a corresponding decoding, which are both expressible in FO+LIN+TCS.*

Proof. We assume that the relation to be encoded involves positive rational numbers only. The general case can be dealt with by splitting the relation into “sign-homogeneous” pieces, dealing with each piece separately and encoding the tuple of natural numbers obtained for each piece again into a single natural number.

In the proof of Lemma 4.2, we have seen that in FO+LIN+TCS we can go from rational numbers (out of a finite set) to denominator/numerator pairs and back. Hence, we can actually assume that the relation to be encoded involves positive natural numbers only.

We will encode this in two steps. In the first step, we encode a finite relation over \mathbf{N} into a finite subset of \mathbf{N} . In the second step, we encode a finite subset of \mathbf{N} into a single natural number. Since queries can be composed, we can treat these two encoding steps (and their corresponding decoding steps) separately.

Encoding, first step. A finite k -ary relation s over \mathbf{N} can be encoded into a finite subset $\text{Enc}_1(s)$ of \mathbf{N} :

$$\text{Enc}_1(s) := \left\{ \prod_{i=1}^k p_i^{n_i} \mid (n_1, \dots, n_k) \in s \right\}.$$

Here, p_i denotes the i th prime number.

Now let S be a k -ary relation name. We will construct an FO+LIN+TC formula ϵ_1 over $\{S\}$ such that for any database D where S^D is finite and involves natural numbers only, $\epsilon_1(D) = \text{Enc}_1(S^D)$. For notational simplicity, we give the construction only for the case $k = 2$; the general case is analogous.

Consider the following formula $\psi(x_1, x_2, y, x'_1, x'_2, y')$:

$$\begin{aligned} \exists u_1 \exists u_2(S(u_1, u_2) \wedge x_1 \leq u_1 \wedge x_2 \leq u_2) \\ \wedge ((x_1 > 0 \wedge x'_1 = x_1 - 1 \wedge x'_2 = x_2 \wedge y' = 2y) \\ \vee (x_1 = 0 \wedge x_2 > 0 \wedge x'_1 = x_1 \wedge x'_2 = x_2 - 1 \wedge y' = 3y)). \end{aligned}$$

Here, $y' = 2y$ is an abbreviation for $y' = y + y$, and similarly for $y' = 3y$; note that 2 and 3 are the first two prime numbers.

We now define the mapping $p(x_1, x_2, y) = (x'_1, x'_2, y)$ if and only if $\psi(x_1, x_2, y, x'_1, x'_2, y')$. As long as $k \leq x_1$, we have that $p^k(x_1, x_2, y) = (x_1 - k, x_2, y2^k)$. As soon as $k > x_1$, $p^k(x_1, x_2, y)$ is undefined. If $k = x_1$, we can compute further iterates and have that $p^{k+\ell}(x_1, x_2, y) = p^\ell(0, x_2 - \ell, y2^{x_1}3^\ell)$ as long as $\ell \leq x_2$. Iterates again become undefined in case $\ell > x_2$. Finally, if $\ell = x_2$ then $p^{k+\ell}(x_1, x_2, y) = (0, 0, y2^{x_1}3^{x_2})$, and we obtain the encoding for (x_1, x_2) for $y = 1$. No further iterates are defined starting from $(0, 0, y')$.

We will compute the iterates of p using transitive closure and check for each (n_1, n_2) whether there exists a k such that $p^k(n_1, n_2, 1) = (0, 0, y)$. More specifically, the desired formula $\epsilon_1(y)$ is equal to

$$\exists n_1 \exists n_2 (S(n_1, n_2) \wedge [\text{TC}_{x_1, x_2, y; x'_1, x'_2, y'} \psi](n_1, n_2, 1, 0, 0, y)).$$

The discussion above shows that this formula gives the correct answer. The condition $S(u_1, u_2) \wedge x_1 \leq u_1 \wedge x_2 \leq u_2$ in ψ bounds the values of x_1 and x_2 , and hence ensures that the transitive closure computation always terminates.

Decoding, first step. Let S be a unary relation name. We will construct an FO+LIN+TC formula δ_1 over $\{S\}$ such that for any database D where S^D equals $\text{Enc}_1(r)$ for some r , we have $\delta_1(D) = r$. As above, we give the construction only for the case $k = 2$.

Consider now the following formula $\psi(x_1, x_2, y, x'_1, x'_2, y')$:

$$x_1 \geq 0 \wedge x_2 \geq 0 \wedge y \geq 1 \wedge ((x'_1 = x_1 + 1 \wedge x'_2 = x_2 \wedge y' = 2y) \vee (x'_1 = x_1 \wedge x'_2 = x_2 + 1 \wedge y' = 3y)) \wedge \exists u(S(u) \wedge y' \leq u).$$

An analysis similar to that for Enc_1 shows that when we define $q(x_1, x_2, y) = (x'_1, x'_2, y')$ if and only if $\psi(x_1, x_2, y, x'_1, x'_2, y')$, the iterates of q satisfy $q^k(0, 0, 1) = (n_1, n_2, u)$ if and only if $u = 2^{n_1}3^{n_2}$.

Then the desired formula $\delta_1(n_1, n_2)$ is

$$\exists u(S(u) \wedge [\text{TC}_{x_1, x_2, y; x'_1, x'_2, y'} \psi](0, 0, 1, n_1, n_2, u)).$$

The condition $\exists u(S(u) \wedge y' \leq u)$ in ψ bounds the value of y' , and hence ensures the termination of the computation of the transitive closure.

Encoding, second step. A finite ordered subset $s = \{n_1, \dots, n_\ell\}$ of \mathbf{N} can be encoded into a single natural number $\text{Enc}_2(s) := \prod_{i=1}^\ell p_i^{n_i}$.

Let S be a unary relation name. We will construct an FO+LIN+TCS formula ϵ_2 over $\{S\}$ such that for any database D where S^D is a finite subset of \mathbf{N} , we have $\epsilon_2(D) = \{\text{Enc}_2(S^D)\}$.

We will use the following auxiliary FO+LIN+TCS formulas; we will explain later how to get them (except for \min and \max , which are easy to get).

- Formulas card , \min , and \max over $\{S\}$, with the property that for any D where S^D is finite of cardinality ℓ , $\text{card}(D) = \{\ell\}$; $\min(D) = \{\min S^D\}$; and $\max(D) = \{\max S^D\}$.
- Formulas prime , mult , and nat , over $\{M\}$, with M a unary relation name, with the property that for any D where $M^D = \{m\}$ is a natural number singleton,
 - $\text{prime}(D) = \{p_m\}$;
 - $\text{mult}(D) = \{(x, y, z) \in \mathbf{R}^3 \mid xy = z \text{ and } y \in \mathbf{N} \text{ and } y \leq m\}$; and
 - $\text{nat}(D) = \{0, 1, 2, \dots, m\}$.

- Formula **pow** over $\{M, M_2\}$, with M, M_2 unary relation names, with the property that for any D where $M^D = \{m\}$ and $M_2^D = \{m_2\}$ are natural number singletons, $\text{pow}(D) = \{(x, y, z) \in \mathbf{R}^3 \mid x^y = z \text{ and } x \in \mathbf{N} \text{ and } x \leq m \text{ and } y \in \mathbf{N} \text{ and } y \leq m_2\}$.

Using composition, we also obtain that

- $\text{maxprime} \equiv \text{prime}(\text{card})$, defining p_ℓ where ℓ is the cardinality of S ;
- $\text{nat}' \equiv \text{nat}(\text{maxprime})$, defining $\{0, 1, 2, \dots, p_\ell\}$; and
- $\text{pow}' \equiv \text{pow}(\text{maxprime}, \text{max})$, defining exponentiation of natural numbers $\leq p_\ell$ by natural numbers $\leq \text{max } S$.

We furthermore construct the following formulas:

- mult' , obtained from mult by replacing each occurrence of a subformula $M(u)$ with

$$\exists p_\ell \exists m (\text{maxprime}(p_\ell) \wedge \text{max}(m) \wedge \text{pow}'(p_\ell, m, u)).$$

This formula defines multiplication by natural numbers $\leq p_\ell^{\text{max } S}$.

- $\text{isprime}(p)$, which defines $\{p_1, p_2, \dots, p_\ell\}$:

$$\text{nat}'(p) \wedge p > 1 \wedge \neg \exists u \exists v (\text{nat}'(u) \wedge \text{nat}'(v) \wedge u > 1 \wedge v > 1 \wedge \text{mult}'(u, v, p)).$$

- $\text{succ}(x, x')$, which specifies the next element after x in S (or $\text{max}(S) + 1$) and is given by the formula

$$\begin{aligned} & (\neg \text{max}(x) \wedge S(x') \wedge x < x' \\ & \wedge \neg \exists x'' (S(x'') \wedge x < x'' < x')) \vee (\text{max}(x) \wedge x' = x + 1). \end{aligned}$$

- $\text{next}(p, p')$, which specifies the next prime number greater than p and smaller than or equal to p_ℓ (or $p_\ell + 1$) and is given by the formula

$$\begin{aligned} & (\neg \text{maxprime}(p) \wedge \text{isprime}(p') \wedge p < p' \\ & \wedge \neg \exists p'' (\text{isprime}(p'') \wedge p < p'' < p')) \vee (\text{maxprime}(p) \wedge p' = p + 1). \end{aligned}$$

We need to compute the product $\prod_{i=1}^{\ell} p_i^{n_i}$. Consider now the following formula $\psi(x, p, y, x', p', y')$:

$$S(x) \wedge \text{succ}(x, x') \wedge \text{next}(p, p') \wedge \exists y'' (\text{pow}'(p, x, y'') \wedge \text{mult}'(y, y'', y')).$$

Note that the variables y and y' are related by $y' = p^x y$. In order to find the desired product, we have to compute the transitive closure of ψ and check which y' -value is in the transitive closure with $(n_1, 2, 1)$ and $(m + 1, p_\ell + 1, y')$. More explicitly, the desired formula $\epsilon_2(n)$ is

$$\begin{aligned} & \exists n_1 \exists m \exists p_\ell (\text{min}(n_1) \wedge \text{max}(m) \wedge \text{maxprime}(p_\ell) \\ & \wedge [\text{TC}_{x,p,y;x',p',y'} \psi](n_1, 2, 1, m + 1, p_\ell + 1, n)). \end{aligned}$$

It remains to show how the auxiliary formulas can be constructed. Formula $\text{card}(\ell)$ can be written as

$$\begin{aligned} & \exists n_1 \exists m (\text{min}(n_1) \wedge \text{max}(m) \\ & \wedge [\text{TC}_{x,c;x',c'} S(x) \wedge \text{succ}(x, x') \wedge c' = c + 1](n_1, 0, m + 1, \ell)), \end{aligned}$$

where $\text{succ}(x, x')$ is as above.

From the computationally completeness of FO+LIN+TCS (Lemma 4.1), we derive directly the formula **prime**.

For formula **mult**, consider the following formula $\psi(x, y, u, x', y', u')$:

$$x' = x \wedge y' = y - 1 \wedge u' = u + x \wedge 0 < y \wedge \exists m(M(m) \wedge y \leq m).$$

Then **mult**(x, y, z) is $[\text{TC}_{x,y,u;x',y',u'} \psi](x, y, 0, x, 0, z)$.

Formula **nat**(n) can be written as

$$n = 0 \vee [\text{TC}_{x;x'} (0 \leq x \wedge \exists m(M(m) \wedge x < m) \wedge x' = x + 1)](0, n).$$

Finally, for formula **pow**, consider the following formula $\psi(x, u, v; x', u', v')$:

$$\begin{aligned} \mathbf{nat}(x) \wedge \exists m(M(m) \wedge x < m) \wedge 0 \leq u \wedge \exists m_2(M_2(m_2) \wedge u < m_2) \wedge u' \\ = u + 1 \wedge \mathbf{mult}(v, x, v'). \end{aligned}$$

Then **pow**(x, y, z) is $(y = 0 \wedge z = 1) \vee [\text{TC}_{x,u,v;x',u',v'} \psi](x, 0, 1, x, y, z)$.

Decoding, second step. Let E be a unary relation name. We will construct an FO+LIN+TCS formula δ_2 over $\{E\}$ such that for any database D where E^D is a singleton $\{e\}$ such that e equals $\text{Enc}_2(s)$ for some s , we have $\delta_2(D) = s$.

By Lemma 4.1, we have formulas **highprime** and **highexp** over $\{E\}$ such that for any D as above, we have **highprime**(D) = $\{p_\ell\}$ and **highexp**(D) = $\{m\}$, where p_ℓ is the highest prime factor of e , and m is the highest exponent of a prime number in the prime factorization of n . Composing the formula **pow** of above with these two formulas, we obtain a formula defining exponentiation of natural numbers $\leq p_\ell$ by natural numbers $\leq m$, which we again denote by **pow'**. Also, analogously to the way we constructed the formula **isprime** above, we obtain a formula defining $\{p_1, p_2, \dots, p_\ell\}$, which we again denote by **isprime**.

We need a formula **divisor** that finds all divisors of a natural number. First, consider the following formula $\psi(u, v, u', v')$:

$$0 \leq u \wedge \exists e(E(e) \wedge u \leq e) \wedge v \geq 1 \wedge v' = v \wedge u' = u - v$$

and let **divisor**(d) be the formula

$$\exists e(E(e) \wedge [\text{TC}_{u,v;u',v'} \psi](e, d, 0, d)).$$

Then, the desired formula $\delta_2(n)$ is

$$\begin{aligned} \exists p(\mathbf{isprime}(p) \wedge \exists d(\mathbf{pow}'(p, n, d) \wedge \mathbf{divisor}(d)) \\ \wedge \neg \exists n' \exists d'(\mathbf{pow}'(p, n', d') \wedge \mathbf{divisor}(d') \wedge n' > n)). \quad \square \end{aligned}$$

4.4. Completeness result for \mathbf{Z} -linear constraint databases.

THEOREM 4.4. *For every partially computable query Q on \mathbf{Z} -linear constraint databases, there exists an FO+LIN+TCS formula φ such that for each database D , $\varphi(D)$ is defined if and only if $Q(D)$ is, and in this case $\varphi(D)$ and $Q(D)$ are equal.*

Proof. The proof follows directly from the lemmas above, as is illustrated in the following diagram. Let D be a \mathbf{Z} -linear constraint database over a schema

$\mathcal{S} = \{S_1, \dots, S_k\}$, and let Q be an arbitrary partially computable query:

$$\begin{array}{ccc}
 D & \xrightarrow{Q} & Q(D) \\
 \downarrow \text{(Lemma 4.2)} & & \uparrow \text{(Lemma 4.2)} \\
 \{S_{1,\text{fin}}, \dots, S_{k,\text{fin}}\} & & S_{\text{fin}} \\
 \downarrow \text{(Lemma 4.3)} & & \uparrow \text{(Lemma 4.3)} \\
 (n_1, \dots, n_k) \in \mathbf{N}^k & \xrightarrow[\text{(Lemma 4.1)}]{f_Q} & n_{Q(D)} \in \mathbf{N}
 \end{array}$$

First, each S_i^D , $i = 1, \dots, k$, is encoded in a finite relations $S_{i,\text{fin}}$, which in its turn is encoded in a natural number n_i . In this way, a k -tuple (n_1, \dots, n_k) is obtained. Since Q is computable, there exists a partial computable function f_Q which implements Q on these encodings. Let $n_{Q(D)}$ be the result of f_Q on input (n_1, \dots, n_k) . This integer is decoded into a finite relation S_{fin} which in its turn is decoded into a \mathbf{Z} -linear constraint database D' . This database is then the result of the query Q on the input database D , i.e., $D' = Q(D)$. \square

4.5. Implications for polynomial constraint databases. For polynomial constraint databases, we cannot prove completeness and have to settle for less. Although finite representations of polynomial constraint databases exist, it is not known whether a finite encoding can be expressed in FO+POLY+TCS.

Let A be a semialgebraic set in \mathbf{R}^n . An *algebraic linearization* of A is an \mathbf{A} -linear set \widehat{A} in \mathbf{R}^n such that A and \widehat{A} are topologically equivalent. A *rational linearization* of A is a \mathbf{Z} -linear set \widehat{A}_{rat} in \mathbf{R}^n such that A and \widehat{A}_{rat} are topologically equivalent.

For $\vec{x} \in \mathbf{R}^n$, we define $\|\vec{x}\| = \sqrt{x_1^2 + \dots + x_n^2}$. A linearization approximates the set A also from a metric point of view if the following condition is satisfied: for every point \vec{p} in A , $\|\vec{p} - h(\vec{p})\| < \varepsilon$ for a fixed $\varepsilon > 0$, where h is a homeomorphism of \mathbf{R}^n such that $h(A) = \widehat{A}$. If this condition is satisfied for a (rational) linearization, we call this linearization a *(rational) ε -approximation* of the set A . We will denote rational and algebraic ε -approximations, respectively, by $\widehat{A}_{\text{rat},\varepsilon}$ and \widehat{A}_ε .

Example 4.1. Consider the planar semialgebraic set $A = \{(x, y) \in \mathbf{R}^2 \mid x^2 + y^2 = 2\}$. Let $\varepsilon = \frac{1}{2}$. In Figure 4.1, we have drawn an algebraic ε -approximation $\widehat{A}_\varepsilon = \{(x, y) \in \mathbf{R}^2 \mid \max\{|x|, |y|\} = \sqrt{2}\}$, a rational ε -approximation $\widehat{A}_{\text{rat},\varepsilon} = \{(x, y) \in \mathbf{R}^2 \mid \max\{|x|, |y|\} = 1\}$, and a linearization \widehat{A} which is not an ε -approximation. \diamond

Algebraic and rational linearizations exist for any semialgebraic set. This is no longer true for ε -approximations, where the existence is guaranteed only for bounded semialgebraic sets. Consider, e.g., the semialgebraic set $\{(x, y) \in \mathbf{R}^2 \mid y = x^2\}$. It is easy to see that this parabola cannot be approximated by a finite number of line segments, and hence has no ε -approximation for any $\varepsilon > 0$.

Let $\mathcal{S} = \{S\}$, with S an n -ary relation name. We define for any polynomial constraint database D over \mathcal{S} an *algebraic (rational) linearization query* Q_{lin} ($Q_{\text{rat-lin}}$) as a query such that $Q_{\text{rat}}(D)$ ($Q_{\text{rat-lin}}(D)$) is an algebraic (rational) linearization of S^D .

Similarly, for any $\varepsilon > 0$ and any polynomial constraint database D over \mathcal{S} such that S^D is a bounded semialgebraic set, we define an *algebraic (rational) ε -approximation query* Q_ε ($Q_{\text{rat},\varepsilon}$) as a query such that $Q_\varepsilon(D)$ ($Q_{\text{rat},\varepsilon}(D)$) is an algebraic (rational) ε -approximation of S^D .

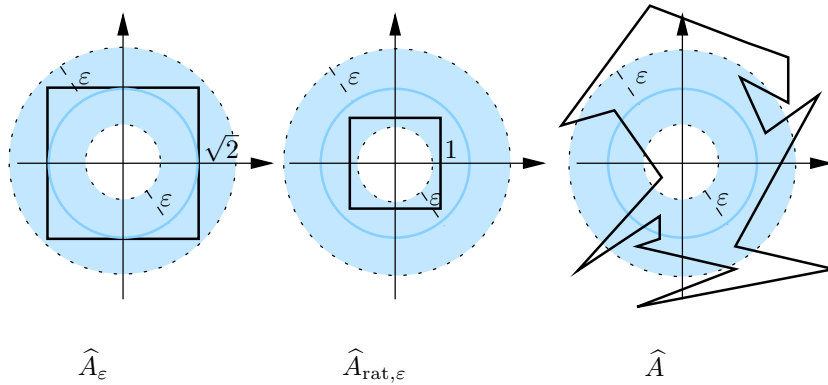


FIG. 4.1. Let A be the circle (grey). Left: an algebraic ε -approximation. Middle: a rational ε -approximation. Right: an algebraic linearization.

It is an open question whether some algebraic or rational linearization query can be expressed in FO+POLY. With respect to the ε -approximation query, neither the algebraic nor the rational version can be expressed in FO+POLY.

PROPOSITION 4.5. Let $\varepsilon > 0$ be a real number. No ε -approximation query is expressible in FO+POLY.

Proof. Let $\mathcal{S} = \{S\}$, with S a binary relation name. Let D be a polynomial constraint database over \mathcal{S} . Consider the following FO+POLY formulas over \mathcal{S} :

- A formula `circle` such that for any database D over \mathcal{S} , `circle`(D) is either the circle through the points of S^D , if S^D consists of three noncollinear points, or `circle`(D) = \emptyset . This formula is easily seen to be in FO+POLY.
- A formula `cornerpoints` such that for any database D over \mathcal{S} , `cornerpoints`(D) is either the set of points in which S^D is not locally a straight line, in the case when S^D is semilinear, or `cornerpoints`(D) = \emptyset , otherwise. By a result of Dumortier et al. [10], it is expressible in FO+POLY whether a semialgebraic set is semilinear. Hence, `cornerpoints` is expressible in FO+POLY.

Assume that the query Q_ε (and similarly, $Q_{\text{rat},\varepsilon}$) is expressible in FO+POLY. Let ε -`approx` be the formula which expresses Q_ε . Then the formula

$$\varphi \equiv \text{cornerpoints}(\varepsilon\text{-approx}(\text{circle}))$$

is also in FO+POLY. However, the number of points in $\varphi(D)$, $|\varphi(D)|$ can be made arbitrarily large by choosing D such that S^D consist of three points far enough apart. This contradicts the dichotomy theorem of Benedikt and Libkin [4], which guarantees the existence of a polynomial p_φ such that $|\varphi(D)| < p_\varphi(|S^D|) = p_\varphi(3)$ in the case when $|\varphi(D)|$ is finite. \square

In contrast to the negative expressiveness result in Proposition 4.5, we will prove that all kinds of linearizations are expressible in FO+POLY+TC. Indeed, in section 6 we show that there exists

- an FO+POLY+TC expressible algebraic linearization query (Theorem 6.6);
- an FO+POLY+TC expressible rational linearization query (Theorem 6.9);
- an FO+POLY+TC expressible algebraic ε -approximation query (Theorem 6.7);
- an FO+POLY+TC expressible rational ε -approximation query (Theorem 6.10).

We shall denote the FO+POLY+TC formula, which expresses the rational linearization by ratlin . Let Q be a partially computable Boolean topological query. Since Q is partially computable, it is in particular partially computable on \mathbf{Z} -linear constraint databases, and therefore by Theorem 4.4 is expressible on these databases by a formula φ_Q in FO+LIN+TCS.

Because Q is topological, $Q(D)$ is true if and only if $\varphi_Q(\text{ratlin}(D))$ is true. Hence, we have proven the following theorem.

THEOREM 4.6. *For every partially computable Boolean topological query Q on polynomial constraint databases, there exists an FO+POLY+TCS formula φ such that for each database D , $\varphi(D)$ is defined if and only if $Q(D)$ is defined, and in this case $\varphi(D)$ and $Q(D)$ are equal.*

5. Geometrical properties of semialgebraic sets. In this section, we discuss a number of topological properties of spatial databases that can be expressed in first-order logic. They are used in the construction of the linearization of polynomial constraint databases in the next section.

We will use the following notation. Let $A \subseteq \mathbf{R}^n$; the closure of A is denoted by $\text{cl}(A)$, and $\text{int}(A)$ indicates the interior of A . We denote $\text{cl}(A) - \text{int}(A)$ (the boundary of A) by ∂A .

5.1. The cone radius. Let A be a semialgebraic set in \mathbf{R}^n , and let \vec{p} be a point in \mathbf{R}^n . We define the *cone with base A and top \vec{p}* as the union of all closed line segments between \vec{p} and points in A . Formally, this is the set $\{t\vec{b} + (1-t)\vec{p} \mid \vec{b} \in A, 0 \leq t \leq 1\}$ and we denote this set by $\text{Cone}(A, \vec{p})$.

For a point $\vec{p} \in \mathbf{R}^n$, and $\varepsilon > 0$, we denote the closed ball centered at \vec{p} with radius ε by $B^n(\vec{p}, \varepsilon)$, and we denote the sphere centered at \vec{p} with radius ε by $S^{n-1}(\vec{p}, \varepsilon)$.

The local conic structure of semialgebraic sets characterizes the local topology of semialgebraic sets.

THEOREM 5.1 (local conic structure; see Theorem 9.3.6 of [6]). *Let A be a semialgebraic set in \mathbf{R}^n and \vec{p} be a point of $\text{cl}(A)$. Then there is a real number $\varepsilon > 0$ such that intersection $B^n(\vec{p}, \varepsilon) \cap A$ is homeomorphic to the set $\text{Cone}(S^{n-1}(\vec{p}, \varepsilon) \cap A, \vec{p})$, in the case when $\vec{p} \in A$, and homeomorphic to $\text{Cone}(S^{n-1}(\vec{p}, \varepsilon) \cap A, \vec{p}) - \{\vec{p}\}$ otherwise.*

Before we can state a “box” version of this theorem, we need the following definitions. Consider a $2n$ -tuple $B = (a_1, b_1, \dots, a_n, b_n) \in \mathbf{R}^{2n}$ with $a_i \leq b_i$ for each i . One can associate with each such tuple an n -ary relation $|B|$ in \mathbf{R}^n :

$$|B| := \{(x_1, \dots, x_n) \in \mathbf{R}^n \mid (a_1 \leq x_1 \leq b_1) \wedge \dots \wedge (a_n \leq x_n \leq b_n)\}.$$

We call B a *box* in \mathbf{R}^n , and $|B|$ is the *geometric realization* of B . The *dimension* of a box is the number of pairs (a_i, b_i) with $a_i \neq b_i$. The *diameter* of a box B , $\text{diam}(B)$, equals $(\sum_{i=1}^n (b_i - a_i)^2)^{1/2}$. The *center* of B is the point $((a_1 + b_1)/2, \dots, (a_n + b_n)/2)$.

THEOREM 5.2 (see [14]). *Let A be a semialgebraic set in \mathbf{R}^n and \vec{p} a point of $\text{cl}(A)$. Then there is a real number $\varepsilon > 0$ such that for any n -dimensional box B in \mathbf{R}^n such that*

1. $\vec{p} \in \text{int}(|B|)$, and
2. $|B| \subseteq (p_1 - \varepsilon, p_1 + \varepsilon) \times \dots \times (p_n - \varepsilon, p_n + \varepsilon)$,

we have that the intersection $A \cap |B|$ is homeomorphic to the set $\text{Cone}(A \cap \partial|B|, \vec{p})$, in the case when $\vec{p} \in A$, and homeomorphic to the set $\text{Cone}(A \cap \partial|B|, \vec{p}) - \{\vec{p}\}$ otherwise.

Any positive real number ε as in Theorem 5.2 is called a *cone radius* of A in \vec{p} (see Figure 5.1).

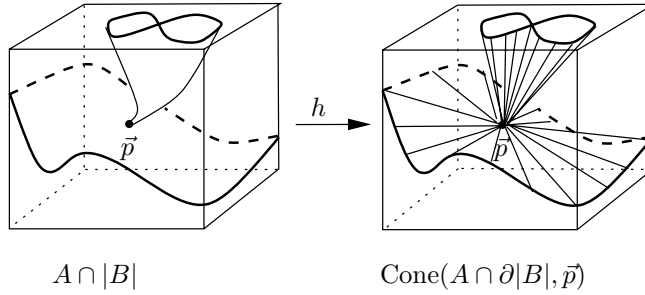


FIG. 5.1. The local conic structure of semialgebraic sets.

Let $\mathcal{S} = \{S\}$, with S an n -ary relation name. We define the *cone radius query* Q_{radius} as a query which maps any polynomial constraint database D over \mathcal{S} to a set of pairs $(\vec{p}, r) \in \mathbf{R}^n \times \mathbf{R}$ such that for every $\vec{p} \in \text{cl}(S^D)$ there exists at least one pair $(\vec{p}, r) \in Q_{\text{radius}}(D)$, and for every $(\vec{p}, r) \in Q_{\text{radius}}(D)$, r is a cone radius of S^D in \vec{p} .

THEOREM 5.3 (see [14]). *The cone radius query defined above is expressible in FO+POLY.*

The FO+POLY formula over \mathcal{S} , constructed in [14] and whose existence is referred to in Theorem 5.3, will be denoted by **radius**. The exact properties of this formula are not important (except for the fact that, for each point \vec{p} , it assigns an open interval $(0, r) \subset \mathbf{R}$ such that for each $r' \in (0, r)$, r' is a cone radius) until the proof of Claim 6.1. There we have to go back to the construction of **radius** for the cone radius query as presented in [14].

As observed above, for each point \vec{p} ,

$$\{r' \mid (\vec{p}, r') \in \text{radius}(D)\} = (0, r).$$

Define $r_{\vec{p}}$ to be the cone radius $r/2$. Moreover, let **uniqueradius** be the FO+POLY formula over \mathcal{S} such that for each point $\vec{p} \in \text{cl}(S^D)$, $(\vec{p}, r_{\vec{p}})$ is in **uniqueradius**(D). Basically, **uniqueradius** assigns a unique cone radius to each point.

For a given semialgebraic set A in \mathbf{R}^n , we now define the semialgebraic mapping² $\gamma_{\text{cone}, A}$ from $\text{cl}(A)$ to \mathbf{R} which maps each point $\vec{p} \in \text{cl}(A)$ to the unique cone radius $r_{\vec{p}} \in \mathbf{R}$ given by **uniqueradius**(D), where $S^D = A$.

5.2. The uniform cone radius decomposition. Although every point of a semialgebraic set has a cone radius which is strictly greater than zero (Theorem 5.2), we are now interested in finding a *uniform cone radius* for a semialgebraic set. We define a uniform cone radius of a semialgebraic set $A \subseteq \mathbf{R}^n$ as a real number $\varepsilon_A > 0$ such that ε_A is a cone radius of A in all points of A . For any $X \subseteq A \subseteq \mathbf{R}^n$, we define a uniform cone radius of X with respect to A as a real number $\varepsilon > 0$ such that ε is a cone radius of A in all points of X .

A first observation is that a uniform cone radius of a semialgebraic set does not always exist.

Example 5.1. Consider the set shown in Figure 5.2. We have drawn the maximal cone radius around the points $\vec{p}_1, \vec{p}_2, \vec{p}_3, \vec{p}_4$, and \vec{p}_5 . It is clear that the closer these points are to the point \vec{p} , the smaller their maximal cone radius is. Because we can make the maximal cone radius arbitrarily small by taking points very close to \vec{p} , we may conclude that the set shown in this figure has no uniform cone radius. \diamond

²A mapping is called semialgebraic if its graph is a semialgebraic set.

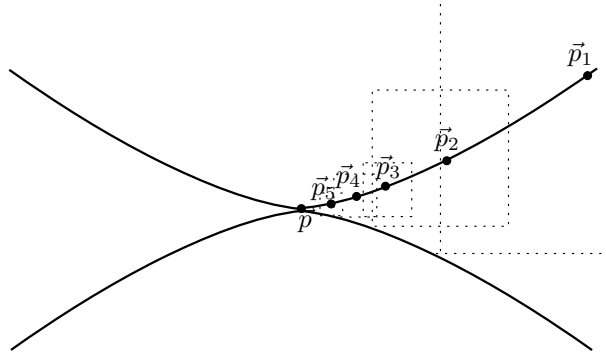


FIG. 5.2. Example of a semialgebraic set which does not have a uniform cone radius.

Let A be a semialgebraic set in \mathbf{R}^n . We define the ε -neighborhood of A as

$$A^\varepsilon := \{\vec{x} \in \mathbf{R}^n \mid \exists \vec{y} (\vec{y} \in A \wedge \|\vec{x} - \vec{y}\| < \varepsilon)\}.$$

We will frequently use the following notation. Let U_0, \dots, U_m be pairwise disjoint semialgebraic subsets of $\text{cl}(A)$, which satisfy the condition that for any m -tuple $(\varepsilon_0, \dots, \varepsilon_m)$ of positive real numbers, and for $i = 0, \dots, m$, the sets

$$(5.1) \quad \inf \left\{ \gamma_{\text{cone}, A} \left(U_i - \bigcup_{j=i+1}^m U_j^{\varepsilon_j} \right) \right\} > 0.$$

Note that these sets have a uniform cone radius with respect to A . Hence, we say that the sets U_0, \dots, U_m form a *uniform cone radius collection* of $\text{cl}(A)$.

When the sets U_0, \dots, U_m of a uniform cone radius collection of A form a decomposition of $\text{cl}(A)$, i.e.,

$$\text{cl}(A) = U_0 \cup \dots \cup U_m,$$

then we call U_0, \dots, U_m a *uniform cone radius decomposition* of $\text{cl}(A)$.

We now show how to construct such a uniform cone radius decomposition of $\text{cl}(A)$. For any closed subset $X \subseteq \text{cl}(A)$, we define

$$(5.2) \quad \Gamma_{\text{nc}}(X) := \{\vec{p} \in X \mid \gamma_{\text{cone}, A} \upharpoonright_X \text{ is not continuous in } \vec{p}\}.$$

Let $\Delta_0 := \text{cl}(A)$, and let $\Delta_{i+1} := \text{cl}(\Gamma_{\text{nc}}(\Delta_i)) \cap \Delta_i$. We define for $k = 0, 1, \dots$ the sets

$$(5.3) \quad C_k := \Delta_k - \Delta_{k+1}.$$

By taking $f = \gamma_{\text{cone}, A}$ in the following lemma, we obtain that $\Gamma_{\text{nc}}(X)$ is semialgebraic and $\dim(\Gamma_{\text{nc}}(X)) < \dim X$.

LEMMA 5.4. *For each semialgebraic set X in \mathbf{R}^n and each semialgebraic function $f : X \rightarrow \mathbf{R}$, the set $\Gamma(f) = \{\vec{p} \in X \mid f(\vec{p}) \text{ is not continuous in } \vec{p}\}$ is semialgebraic and $\dim(\Gamma(f)) < \dim X$.*

Proof. The set

$$\Gamma(f) = \{\vec{p} \in \mathbf{R}^n \mid (\exists \varepsilon > 0)(\forall \delta > 0)\exists \vec{q} \in \mathbf{R}^n (\vec{q} \in X \cap B^n(\vec{p}, \delta) \wedge |f(\vec{p}) - f(\vec{q})| > \varepsilon)\}$$

is clearly semialgebraic. This proves the first assertion.

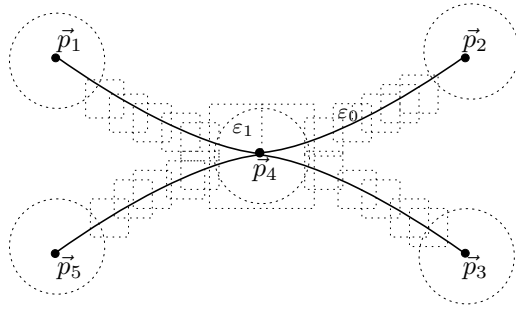


FIG. 5.3. The points $\vec{p}_1, \vec{p}_2, \vec{p}_3, \vec{p}_4,$ and \vec{p}_5 form the part C_1 which has ε_1 as uniform cone radius. As can be seen, the set $C_0 = A - C_1^{\varepsilon_1}$ has a uniform cone radius ε_0 .

We prove the second assertion by contradiction. Let $d = \dim X$ and suppose that $\dim(\Gamma(f)) = d$. Then there exists a semialgebraic cell $V \subseteq \Gamma(f)$ of dimension d . By the cell decomposition theorem of semialgebraic sets [44, Theorem 2.11] there exists a semialgebraic cell decomposition of V into a finite number of semialgebraic cells,

$$V = V_1 \cup \dots \cup V_k \cup V_{k+1} \cup \dots \cup V_\ell,$$

with $\dim(V_i) = d$ for $i = 1, \dots, k$ and $\dim(V_j) < d$ for $j = k + 1, \dots, \ell$, such that

$$(5.4) \quad f|_{V_i} \text{ is continuous for every } i = 1, \dots, \ell.$$

Since $V_i \subseteq V$ has dimension d for $i = 1, \dots, k$, V_i is open in V , and V_i is also open in X for $i = 1, \dots, k$. From (5.4) we deduce that each V_i for $i = 1, \dots, k$ is included in $X - \Gamma(f)$, which is impossible since $V \subseteq \Gamma(f)$. Hence, $\dim(\Gamma(f)) < d$. \square

An immediate consequence of this lemma is that from $i = n + 1$ on, the C_i 's are all empty. Let us denote by m the latest index such that C_m is nonempty. Thus, $m \leq n$.

We now prove that for any tuple $(\varepsilon_0, \dots, \varepsilon_m)$ of positive real numbers, the sets

$$C_i - \bigcup_{j=i+1}^m C_j^{\varepsilon_j} \quad \text{for } i = 0, 1, \dots, m$$

have a uniform cone radius. Since $C_m = \Delta_m$ is closed, $\gamma_{\text{cone}, A}(C_m)$ is also closed and therefore has a minimum which is strictly positive. Hence, C_m has a uniform cone radius. For $i > 0$ there exists an $\eta < \min\{\varepsilon_0, \dots, \varepsilon_m\}$ such that

$$(5.5) \quad C_i - \bigcup_{j=i+1}^m C_j^{\varepsilon_j} \subseteq Z := \Delta_i - \Delta_{i+1}^\eta.$$

The set Z is closed and the restriction $\gamma_{\text{cone}, A}|_Z$ is continuous. Hence, $\gamma_{\text{cone}, A}(Z)$ is closed in \mathbf{R} and has a minimum which is strictly positive. We may conclude that Z has a uniform cone radius, and by (5.5), so has $C_i - \bigcup_{j=i+1}^m C_j^{\varepsilon_j}$. Thus, C_0, \dots, C_m is a uniform cone radius decomposition of $\text{cl}(A)$.

Example 5.2. In Figure 5.3, we have shown the uniform cone radius decomposition of the set depicted in Figure 5.2. \diamond

Let $\mathcal{S} = \{S\}$, with S an n -ary relation name. We define the $n + 1$ queries Q_k^{uniform} such that for any polynomial constraint database D over \mathcal{S} ,

$$Q_k^{\text{uniform}}(D) := C_k$$

for $k = 0, 1, \dots, n$, with C_0, \dots, C_n being the uniform cone radius decomposition of $\text{cl}(S^D)$.

Because $\gamma_{\text{cone}, S^D}$ equals $\text{uniqueradius}(D)$, and by Theorem 5.3 the formula uniqueradius is in FO+POLY, the following lemma is immediate.

LEMMA 5.5. *The queries $Q_{k\text{-uniform}}$, $k = 0, 1, \dots, n$, are expressible in FO+POLY.*

5.3. The regular decomposition. In this section, we construct a decomposition of semialgebraic sets such that a certain regularity condition is satisfied on each part of the decomposition. In order to define this regularity condition, we need to define the tangent space to a semialgebraic set in a point. The following definitions are taken from Rannou [39].

Let A be a semialgebraic set in \mathbf{R}^n . The *secants limit set* of A in a point $\vec{p} \in A$ is defined as the set

$$\text{limsec}_{\vec{p}} A := \bigcap_{\eta > 0} \text{cl}(\{\lambda(\vec{u} - \vec{v}) \in \mathbf{R}^n \mid \lambda \in \mathbf{R} \text{ and } \vec{u}, \vec{v} \in A \cap B^n(\vec{p}, \eta)\}).$$

If $\text{limsec}_{\vec{p}} A$ is a vector space, then we define the *tangent space of A in \vec{p}* as $T_{\vec{p}} A := \vec{p} + \text{limsec}_{\vec{p}} A$. If $\text{limsec}_{\vec{p}} A$ is not a vector space, then the tangent space of A in \vec{p} is undefined.

Let $\mathcal{S} = \{S\}$, with S an n -ary relation name. We define the query Q_{tangent} as the query such that for any polynomial constraint database D over \mathcal{S} ,

$$Q_{\text{tangent}}(D) := \{(\vec{x}, \vec{v}) \in S^D \times \mathbf{R}^n \mid T_{\vec{x}} S^D \text{ exists in } \vec{x} \text{ and } \vec{v} \in T_{\vec{x}} S^D\}.$$

LEMMA 5.6. *The query Q_{tangent} is expressible in FO+POLY.*

Proof. It is shown by Rannou [39, Lemma 2] that the definition of the secant limit set of a set in a point can be translated into a first-order formula over the reals. Since it is straightforward to check in FO+POLY whether a secant limit set is a vector space (i.e., we need to check whether, for all \vec{s}, \vec{t} in a secant limit set, the sum $\vec{s} + \vec{t}$ is also an element of this secant limit set), the lemma is immediate. \square

The set A is *regular in \vec{p}* if and only if $T_{\vec{p}} A$ exists and there exists a neighborhood U of \vec{p} such that the orthogonal projection of $A \cap U$ on $T_{\vec{p}} A$ is bijective. A set is *regular* if it is regular in all of its points.

A finite number of pairwise disjoint regular sets R_1, \dots, R_k is called a *regular decomposition of A* if $A = R_1 \cup \dots \cup R_k$.

We now show that every semialgebraic set A has a regular decomposition.

We denote the set of points where A is regular and where the local dimension of A is k by $\text{Reg}_k(A)$. Note that $\text{Reg}_k(A)$ is either empty or $\dim \text{Reg}_k(A) = k$.

Define inductively for $k = n, n - 1, \dots, 0$, the sets

$$(5.6) \quad R_k := \text{Reg}_k \left(A - \bigcup_{j=k+1}^n R_j \right).$$

These sets are pairwise disjoint and form a decomposition of A , i.e.,

$$(5.7) \quad A = R_n \cup R_{n-1} \cup \dots \cup R_0.$$

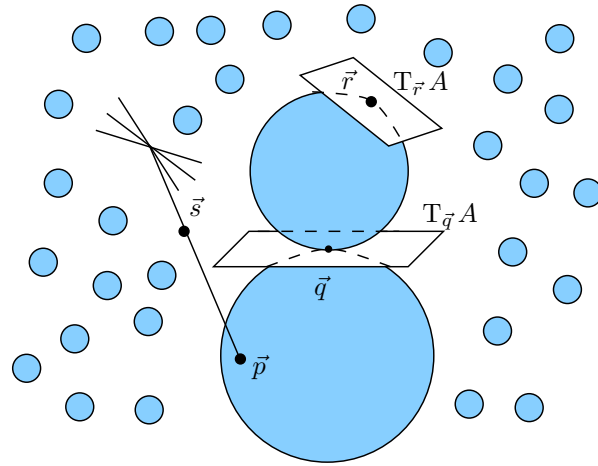


FIG. 5.4. The snowman A has no tangent space in \vec{p} , A has a tangent space in \vec{q} and \vec{r} but is not regular in these points, and A is regular in \vec{s} .

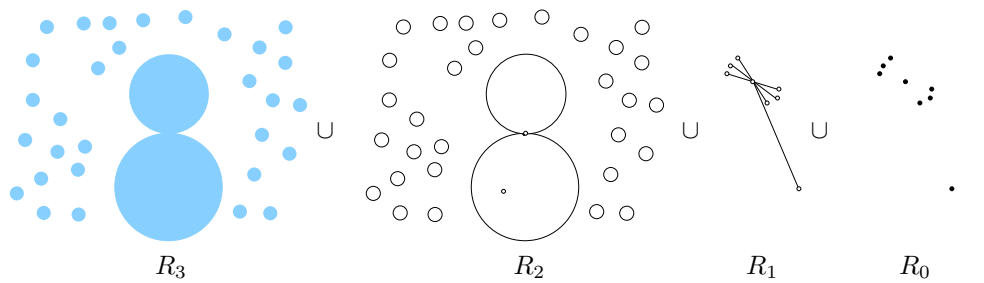


FIG. 5.5. The three-dimensional set A of Figure 5.4 is decomposed into four parts R_0 , R_1 , R_2 , and R_3 according to the construction of the regular decomposition.

Note that $n + 1$ parts are really sufficient because, for any semialgebraic set $X \subseteq \mathbf{R}^n$ of dimension d , $X - \text{Reg}_d(X)$ has a strictly lower dimension than X [45].

Moreover, by (5.6) each R_k is regular, and hence we define the *regular decomposition* of A as the $n + 1$ sets R_0, \dots, R_n .

Example 5.3. In Figure 5.4, we have illustrated the three possible cases: $T_{\vec{p}}A$ does not exist; $T_{\vec{q}}A$ and $T_{\vec{r}}A$ exist but A is not regular in \vec{q} and \vec{r} ; and finally, A is regular in \vec{s} . In Figure 5.5, we have drawn an example of the regular decomposition of a three-dimensional set in \mathbf{R}^3 . \diamond

Let $\mathcal{S} = \{S\}$, with S an n -ary relation name. We define the $n + 1$ queries Q_k^{reg} as the queries such that for every polynomial constraint database D ,

$$Q_k^{\text{reg}}(D) := R_k$$

for $k = 0, \dots, n$, with R_0, \dots, R_n the regular decomposition of S^D .

It was proved by Rannou [39, Proposition 2] that checking whether a semi-algebraic set is regular in a point is first-order expressible. Hence the next lemma follows.

LEMMA 5.7. *The queries $Q_{k\text{-reg}}$, $k = 0, 1, \dots, n$ are expressible in FO+POLY.*

Regular decompositions of semilinear sets are fully treated by Dumortier et al. [10] and Vandeuren [46]. These authors showed that on semilinear databases, the $n + 1$

queries $Q_{k\text{-reg}}$ are already expressible in FO+LIN. There is, however, a great difference between the semilinear and semialgebraic cases. Indeed, in the semialgebraic case, regularity implies that the set is a C^1 -smooth algebraic variety, while in the semilinear case, regularity implies that the set is a C^∞ -smooth algebraic variety. One could ask if it is possible to define a regularity condition in first-order logic such that it also induces C^∞ -smoothness of semialgebraic sets, but this is impossible [49].

However, we still can generalize the regular decompositions defined above to C^k -regular decompositions by demanding C^k -smoothness instead of C^1 -smoothness (regularity). Using again results from Rannou [39, Proposition 3], we have first-order expressibility of the corresponding query in this case too.

An interesting question is which extensions of FO+POLY can express C^∞ -regular decompositions. A useful observation in this context might be that for every semialgebraic set, there exists a natural number K such that for all $k > K$, a C^k -regular decomposition is already a C^∞ -regular decomposition. Unfortunately, it is not known how to find K for a given semialgebraic set [40] and we might have to compute C^k -regular decompositions for increasing values of k until two consecutive decompositions are identical. This indicates that recursion is needed for the computation of C^∞ -regular decompositions. We leave open whether the recursion in FO+POLY+TC or FO+POLY+TCS is sufficient for this purpose.

5.4. Transversality. In computational geometry [9], a convenient assumption is the hypothesis of “general position,” which dispenses with the detailed consideration of special cases. In the description of our linearization algorithm in section 6, we would like to assume this hypothesis. However, we need to make precise what we will mean by general position and see if this hypothesis may indeed be assumed.

Let A and B be two regular semialgebraic sets in \mathbf{R}^n . From differential topology [23], we recall that A and B are said to *intersect transversally* at $\vec{p} \in A \cap B$ if³

$$(5.8) \quad T_{\vec{p}}A + T_{\vec{p}}B = \mathbf{R}^n.$$

The sets A and B are *in general position* if they intersect transversally in every point of $A \cap B$. We denote this by $A \pitchfork B$. This is illustrated in Figure 5.6 and Figure 5.7, where some examples of transversal and nontransversal intersections in \mathbf{R}^2 and \mathbf{R}^3 are depicted.

Let $\mathcal{A} = \{A_1, \dots, A_n\}$ and $\mathcal{B} = \{B_1, \dots, B_m\}$ be finite sets of regular semialgebraic sets in \mathbf{R}^n such that $A_i \cap A_j = \emptyset$ and $B_i \cap B_j = \emptyset$ for $i \neq j$. We say that \mathcal{A} and \mathcal{B} are in general position if A_i and B_j are in general position for every $i = 1, \dots, n$ and every $j = 1, \dots, m$. We denote this by $\mathcal{A} \pitchfork \mathcal{B}$.

Let $\mathcal{S} = \{S_1, S_2\}$, with S_1 and S_2 two n -ary relation names. We define the Boolean query Q_{\pitchfork} such that for every polynomial constraint database D over \mathcal{S} ,

$$Q_{\pitchfork}(D) = \text{true} \text{ if and only if } S_1^D \text{ and } S_2^D \text{ are regular and } S_1^D \pitchfork S_2^D.$$

Condition (5.8) can be readily expressed in FO+POLY, and by Lemma 5.7, regularity is expressible in FO+POLY. Hence Lemma 5.8 follows.

LEMMA 5.8. *The Boolean query Q_{\pitchfork} is expressible in FO+POLY.*

Given two arbitrary regular semialgebraic sets A and B in \mathbf{R}^n not in general position, we can ask how to force them to be in general position. The following

³Let U and V be two subspaces of a vector space X ; then the *sum* $U + V$ is the set of all vectors $\vec{u} + \vec{v}$, where $\vec{u} \in U$ and $\vec{v} \in V$. Besides, $U + V$ is a subspace of X .

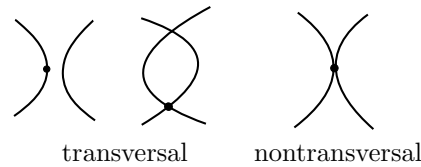


FIG. 5.6. Curves in \mathbf{R}^2 .

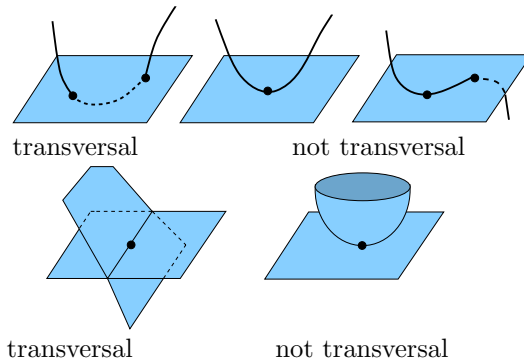


FIG. 5.7. Curves and surfaces in \mathbf{R}^3 .

theorem answers this question. A translation of a set $X \subseteq \mathbf{R}^n$ is a set of the form $X + \tau := \{\vec{x} + \tau \in \mathbf{R}^n \mid \vec{x} \in X\}$, where $\tau \in \mathbf{R}^n$.

THEOREM 5.9. *Let A and B two regular semialgebraic sets in \mathbf{R}^n . For almost all $\tau \in \mathbf{R}^n$, we have that $A + \tau$ and B are in general position.*

Proof. This theorem is a direct consequence of the transversality theorem of differential topology. A proof of the transversality theorem given by Guillemin and Pollack [23] for C^∞ -smooth varieties in \mathbf{R}^n literally remains valid in this case, except that the C^1 -version of Sard’s theorem given by Wilkie [50] needs to be used instead of the standard C^∞ -smooth version. \square

Here, “almost all” means that the set of translation vectors τ for which $A + \tau$ and B are not in general position has *measure zero*.⁴ Since a set of measure zero cannot contain an open set in \mathbf{R}^n , the set of translation vectors τ for which $A + \tau$ and B are in general position is dense in \mathbf{R}^n .

Moreover, Theorem 5.9 can be easily generalized as follows.

COROLLARY 5.10. *Let $\mathcal{A} = \{A_1, \dots, A_n\}$ and $\mathcal{B} = \{B_1, \dots, B_m\}$ be sets of regular semialgebraic sets in \mathbf{R}^n such that $A_i \cap A_j = \emptyset$ ($B_i \cap B_j = \emptyset$) for $i \neq j$. Then for almost all $\tau \in \mathbf{R}^n$, $\mathcal{A} + \tau \pitchfork \mathcal{B}$.*

We mention three useful properties of sets in general position. Let \mathcal{A} and \mathcal{B} be as above; then if $\mathcal{A} \pitchfork \mathcal{B}$, there exists an $\varepsilon > 0$ such that $\mathcal{A} + \tau \pitchfork \mathcal{B}$ for any $\tau \in \mathbf{R}^n$ of norm less than ε . Therefore, one says that transversality is a *stable* property. A second useful property is that the intersection of two regular sets in general position is again regular. A third property is that the tangent space in a point of the intersection of two sets in general position is the intersection of the tangent spaces of these sets in this point [23].

⁴A set in \mathbf{R}^n has *measure zero* if it can be covered by a countable number of n -dimensional boxes with arbitrary small volume.

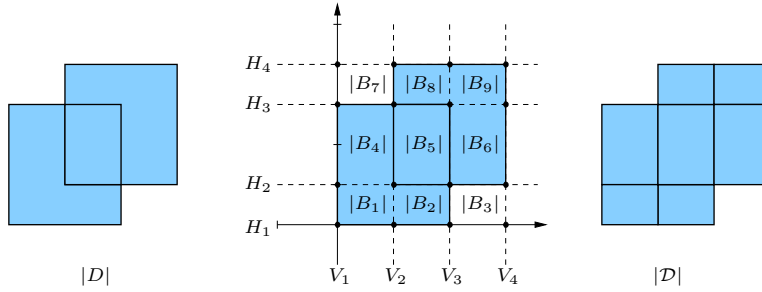


FIG. 5.8. A two-dimensional example of the construction of a box collection for two boxes in the \mathbf{R}^2 .

5.5. Box collections. We need one more ingredient before we can start explaining the linearization algorithm: box collections.

We define an n -dimensional box collection \mathcal{B} in \mathbf{R}^n as a finite set of n -dimensional boxes satisfying an intersection condition. Let B_1 and B_2 be two arbitrary boxes in \mathcal{B} . Then, if $|B_1|$ and $|B_2|$ intersect, the intersection is included in their boundaries $\partial|B_1|$ and $\partial|B_2|$. By the *geometric realization* $|\mathcal{B}|$ of \mathcal{B} , we mean the union of the geometric realizations of all boxes in \mathcal{B} . If $X \subseteq \mathbf{R}^n$ is a semialgebraic set and \mathcal{B} an n -dimensional box collection in \mathbf{R}^n , then $\mathcal{B} \cap X$ is the set of all boxes $B \in \mathcal{B}$ such that $B \cap X \neq \emptyset$.

Let D be a set of n -dimensional boxes, which does not necessarily satisfy the above intersection condition. In the following, we show how in FO+POLY to split the boxes in D into smaller boxes such that the collection of these smaller boxes is a box collection. We call this the *box collection of D* and denote it by \mathcal{D} . By construction, the geometric realization of each box in D is the union of the geometric realizations of certain boxes of \mathcal{D} .

We first give an example of the construction and then present the general construction more formally.

Example 5.4. Fix the dimension $n = 2$ and consider the set D consisting of two boxes $(0, 2, 0, 3)$ and $(1, 3, 1, 4)$. The geometric realization $|D|$ of D is depicted in Figure 5.8. In this figure, two sets of lines, $\mathcal{H}_{D,x} = \{H_1, H_2, H_3, H_4\}$ and $\mathcal{H}_{D,y} = \{V_1, V_2, V_3, V_4\}$, are drawn. Denote the intersection $\bigcup \mathcal{H}_{D,x} \cap \bigcup \mathcal{H}_{D,y}$ by I . In this example, I consists of 16 points $\{\bar{p}_1, \dots, \bar{p}_{16}\}$. From these points, we construct the set \mathcal{P} which contains the 9 two-dimensional boxes denoted by $B_i, i = 1, \dots, 9$. The geometric realizations of these boxes are shown in the figure. As can be seen, these boxes intersect only at their boundaries, and hence form a two-dimensional box collection. Finally, we define the box collection \mathcal{D} of D as the boxes included in $|D|$, i.e., $\mathcal{D} = \{B_1, B_2, B_4, B_6, B_8, B_9\}$. \diamond

In general, we define n unions of $(n - 1)$ -dimensional hyperplanes,

$$\mathcal{H}_{D,i} := \{(x_1, \dots, x_n) \in \mathbf{R}^n \mid \exists (a_1, b_1, \dots, a_n, b_n) \in D \wedge (x_i = a_i \vee x_i = b_i)\},$$

for $i = 1, \dots, n$. Let $I \subseteq \mathbf{R}^n$ be the set of points $\mathcal{H}_{D,1} \cap \dots \cap \mathcal{H}_{D,n}$.

It is easily shown that I is a finite set of points. Indeed, a proof by induction shows that $\dim(\mathcal{H}_{D,1} \cap \dots \cap \mathcal{H}_{D,k}) = n - k$ for any $k = 1, \dots, n$. In particular, $\dim(I) = n - n = 0$, or in other words, I is a finite set.

Next, we construct an n -dimensional box collection, which we denote by \mathcal{P} , such that the geometric realization of each box in D is the union of the geometric

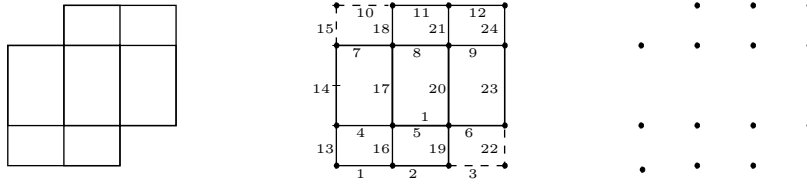


FIG. 5.9. The set $|\mathcal{D}| - |\mathcal{D}|_2$ (left). The one-dimensional box collection $\mathcal{P}_x \cup \mathcal{P}_y$, where the line segment L_i is labeled with the number i (center). The set $|\mathcal{D}|_0$ (right).

realizations of boxes in \mathcal{P} . More specifically,

$$\mathcal{P} := \left\{ (a_1, b_1, \dots, a_n, b_n) \in \mathbf{R}^{2n} \mid \exists \vec{p}_1 \exists \vec{q}_1 \dots \exists \vec{p}_n \exists \vec{q}_n \in I \right. \\ \left. \bigwedge_{i=1}^n (a_i = (\vec{p}_i)_i \wedge b_i = (\vec{q}_i)_i \wedge a_i < b_i) \right. \\ \left. \wedge \left(\forall \vec{r} \in I \bigwedge_{i=1}^n \neg (a_i < (\vec{r})_i < b_i) \right) \right\}.$$

Finally, we define \mathcal{D} as those n -dimensional boxes B in \mathcal{P} such that $|B|$ is included in the geometric realization of any of the boxes in \mathcal{B} . By construction, \mathcal{D} is a box collection, and the geometric realization of any box in \mathcal{D} is the union of the geometric realizations of certain boxes in \mathcal{B} . The construction of \mathcal{D} for a given D can be expressed in FO+POLY, as is clear from the above expressions for $\mathcal{H}_{D,i}$ and \mathcal{P} .

Let $\mathcal{S} = \{S\}$, with S a $2n$ -ary relation name. We define the *box collection query* Q_{bc} such that for any polynomial constraint database D over \mathcal{S} representing a set of n -dimensional boxes in \mathbf{R}^n ,

$$Q_{bc}(D) = \mathcal{D},$$

where \mathcal{D} is the box collection of D . From the constructions above, the following result is immediate.

LEMMA 5.11. *The query Q_{bc} is expressible in FO+POLY.*

When applied to the union of two box collections D and D' , we will denote the box collection $Q_{bc}(D \cup D')$ by $\mathcal{D} \sqcup \mathcal{D}'$.

We next define a useful decomposition of box collections. We again give first an example.

Example 5.5 (see Figure 5.8 and Figure 5.9). Let us continue the previous example. Let $|\mathcal{D}|_2$ be the set in \mathbf{R}^2 defined by $\bigcup_{i \in \{1,2,4,5,6,8,9\}} \text{int}(|B_i|)$. Consider the set $|\mathcal{D}| - |\mathcal{D}|_2$ and define \mathcal{P}_x to be the set of horizontal line segments L_i , with $i = 1, \dots, 12$, and let \mathcal{P}_y be the set of vertical line segments L_i , with $i = 13, \dots, 24$. The line segments L_i can easily be defined from the points in I and from a one-dimensional box collection. We define \mathcal{D}^1 to be the box collection consisting of boxes in $\mathcal{P}_x \cup \mathcal{P}_y$, which are contained in $|\mathcal{D}|$. Next, define $|\mathcal{D}|_1$ to be the set $\bigcup_{i \in \{1, \dots, 24\} - \{3, 10, 22, 15\}} \text{int}(|L_i|)$. Here, when taking the interior, we regard each $|L_i|$ as a space on itself, so the result is an open line segment without the endpoints (as opposed to the empty set, where we would regard each $|L_i|$ as a set in \mathbf{R}^2). Now, $|\mathcal{D}| - |\mathcal{D}|_2 - |\mathcal{D}|_1$ is a subset of I , which we denote by $|\mathcal{D}|_0$. Hence, we have obtained a decomposition of $|\mathcal{D}|$. \diamond

This decomposition is important for two reasons. First, the geometric realization of each box of \mathcal{D} is the disjoint union of the interiors of the geometric realizations of certain boxes in \mathcal{D}^2 , \mathcal{D}^1 , and \mathcal{D}^0 . Second, the interiors of boxes in \mathcal{D} are open subsets of $Reg_2(|\mathcal{D}|)$, the interiors of boxes in \mathcal{D}^1 are open subsets of $Reg_1(|\mathcal{D}| - |\mathcal{D}|_2)$, and finally, $|\mathcal{D}|_0$ equals $Reg_0(|\mathcal{D}| - |\mathcal{D}|_2 - |\mathcal{D}|_1)$.

In general, the construction of this decomposition goes as follows. For $k = 0, 1, \dots, n$ and any combination of k different elements i_1, \dots, i_k in $\{1, \dots, n\}$, we define the following set of $(n - k)$ -dimensional boxes in \mathbf{R}^n :

$$(5.9) \quad \mathcal{P}_{\{i_1, \dots, i_k\}} := \left\{ (a_1, b_1, \dots, a_n, b_n) \in \mathbf{R}^{2n} \mid \exists \vec{p}_1 \exists \vec{q}_1 \dots \exists \vec{p}_n \exists \vec{q}_n \in I \right. \\ \left. \bigwedge_{i \in \{1, \dots, n\}} (a_i = (\vec{p}_i)_i \wedge b_i = (\vec{q}_i)_i) \wedge \forall \vec{r} \in I \bigwedge_{i=1}^n \neg(a_i < (\vec{r})_i < b_i) \right. \\ \left. \wedge \bigwedge_{i \in \{1, \dots, n\} - \{i_1, \dots, i_k\}} a_i < b_i \wedge \bigwedge_{i \in \{i_1, \dots, i_k\}} a_i = b_i \right\}.$$

Note that $\mathcal{P}_{\{1, \dots, n\}} = I$ and $\mathcal{P}_\emptyset = \mathcal{P}$. It is clear that these sets are expressible in FO+POLY. We also define for $k = 0, 1, \dots, n$ and any combination of k different elements i_1, \dots, i_k in $\{1, \dots, n\}$ the following $(n - k)$ -dimensional box collection in \mathbf{R}^n :

$$\mathcal{D}_{\{i_1, \dots, i_k\}} := \left\{ (a_1, b_1, \dots, a_n, b_n) \in \mathcal{P}_{\{i_1, \dots, i_k\}} \mid \exists (a'_1, b'_1, \dots, a'_n, b'_n) \in \mathcal{D} \right. \\ \left. \wedge \bigwedge_{i=1}^n (a'_i \leq a_i \wedge b_i \leq b'_i) \right\}.$$

We then define

$$\mathcal{D}^{n-k} := \bigcup_{\{i_1, \dots, i_k\}} \mathcal{D}_{\{i_1, \dots, i_k\}}.$$

Finally, for $k = 0, 1, \dots, n$, we define $|\mathcal{D}|_{n-k}$ as the union of the interiors of the geometric realizations of boxes in \mathcal{D}^{n-k} . Here, when taking the interior, we regard each geometric realization of a box as a space on itself, so the result is an open box. By construction, we have the following properties:

1.

$$(5.10) \quad |\mathcal{D}| = |\mathcal{D}|_n \cup \dots \cup |\mathcal{D}|_0;$$

2. each geometric realization of a box in \mathcal{D} is the union of the geometric realizations of boxes in $|\mathcal{D}|_k$ for $k = 0, 1, \dots, n$; and
3. the interiors of the geometric realizations of boxes in \mathcal{D}^k are open subsets of $Reg_k(|\mathcal{D}| - |\mathcal{D}|_n - \dots - |\mathcal{D}|_{k+1})$.

Let $\mathcal{S} = \{S\}$, with S a $2n$ -ary relation name. We define the $n + 1$ queries $Q_{k\text{-box}}$ such that for any polynomial constraint database D over \mathcal{S} representing a box collection \mathcal{D} ,

$$Q_{k\text{-box}}(D) = \mathcal{D}^i$$

for $k = 0, 1, \dots, n$ with \mathcal{D}^i the i -dimensional box collection in \mathbf{R}^n defined above. The following trivially holds.

LEMMA 5.12. *The queries $Q_{k\text{-box}}$, $k = 0, 1, \dots, n$, are expressible in FO+POLY.*

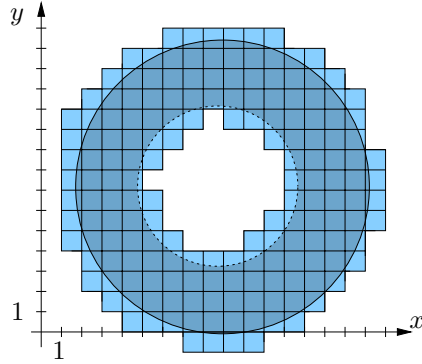


FIG. 5.10. The δ -cover of a semiopen annulus for $\delta = 1$.

5.6. Expressing the box covering query. Let $\delta > 0$ be a real number. We define the n -dimensional standard grid of size δ , called the δ -grid, as the n -dimensional box collection δ -grid consisting of all boxes of the form $(k_1\delta, (k_1 + 1)\delta, \dots, k_n\delta, (k_n + 1)\delta)$, where $k_1, \dots, k_n \in \mathbf{Z}$. We define the *box covering of size δ* of a semialgebraic set A , denoted by $\delta\text{-cover}(A)$, as those boxes in the δ -grid that intersect the closure of A (see Figure 5.10). Let $\mathcal{S} = \{S\}$, with S an n -ary relation name. For each $\delta > 0$ we define the *box covering query* $Q_{\delta\text{-cover}}$ such that for every constraint database D over \mathcal{S} ,

$$Q_{\delta\text{-cover}}(D) := \delta\text{-cover}(S^D).$$

PROPOSITION 5.13. *Let $\delta > 0$. The query $Q_{\delta\text{-cover}}$ is not expressible in FO+POLY.*

Proof. Let $\mathcal{S} = \{S\}$, with S a binary relation name. We consider the following FO+POLY formula over \mathcal{S} : a formula `circle` such that for any database D over \mathcal{S} , either `circle`(D) is the circle through the points of S^D , if S^D consists of three noncollinear points, or `circle`(D) = S^D .

Assume that the query $Q_{\delta\text{-cover}}$ is expressible in FO+POLY. Let $\delta\text{-cover}$ be the formula which expresses $Q_{\delta\text{-cover}}$. Then the formula

$$\varphi \equiv \delta\text{-cover}(\text{circle})$$

is also expressible in FO+POLY. However, the number of 4-tuples in $\varphi(D)$ can be made arbitrarily large by choosing D to be a database over \mathcal{S} such that S^D consists of three points far enough apart. This contradicts the dichotomy theorem of Benedikt and Libkin [4], which guarantees the existence of a polynomial p_φ such that $|\varphi(D)| < p_\varphi(|S^D|) = p_\varphi(3)$ in the case when $|\varphi(D)|$ is finite. \square

However, in FO+POLY+TC we can express the box covering query as follows.

PROPOSITION 5.14. *For each $\delta > 0$, the query $Q_{\delta\text{-cover}}$ is expressible in FO+POLY+TC when restricted to bounded input databases.*

Proof. Let $\mathcal{S} = \{S\}$, with S an n -ary relation name. We define the bounding box query Q_{bb} as the query such that for every polynomial constraint database D , such that S^D is bounded, $Q_{\text{bb}}(D) := \{M\}$, with M a real number such that $\text{cl}(S^D) \subseteq [-M, M]^n$. This query is clearly FO+POLY expressible by a formula over \mathcal{S} which we denote by `boundingbox`(x). Let

$$\begin{aligned} \text{grid}(u) \equiv & [\text{TC}_{x;x'} \exists M (\text{boundingbox}(M) \wedge x \geq 0 \\ & \wedge x' = x + \delta \wedge x' \leq M)](0, u) \vee u = 0. \end{aligned}$$

Let

$$\delta\text{-cover}(u_1, v_1, \dots, u_n, v_n) \equiv \bigwedge_{i=1}^n (v_i = u_i + \delta \wedge \text{grid}(u_i)) \\ \wedge \exists \vec{x} \left(\text{cl}(S)(\vec{x}) \wedge \bigwedge_{i=1}^n u_i < x_i < v_i \right).$$

Then $Q_{\delta\text{-cover}}(D) = \delta\text{-cover}(D)$ for any database D over \mathcal{S} such that S^D is bounded. \square

6. Linearization and approximation of semialgebraic sets. In this section, we give a construction of both an algebraic linearization and an ε -approximation of semialgebraic sets which are implementable in FO+POLY+TC. This implementation is based on the construction of a box collection satisfying some special properties.

More specifically, it is shown in section 6.1 how to construct such a box collection \mathcal{R} for a semialgebraic set A . In section 6.2 we derive a box collection \mathcal{U} from \mathcal{R} and take a closer look at A on the boundaries of \mathcal{U} . We show that we can apply the construction in section 6.1 again for A on the lower-dimensional box collections on the boundaries of \mathcal{U} . This inductive process is the basis of the algorithm LINEARIZE in section 6.3 which builds an algebraic linearization and an ε -approximation of bounded semi-algebraic sets. In the same section, we prove the correctness of the algorithm LINEARIZE and show that the algorithm can be expressed by a query in FO+POLY+TC.

We also show how to extend this algorithm such that it also builds algebraic linearizations of possibly unbounded semialgebraic sets. Finally, in section 6.4 we show that after some minor changes, the algorithm LINEARIZE can be used to build a rational linearization and an ε -approximation of semialgebraic sets.

6.1. Construction of a special box collection. Let \mathcal{B} be an n -dimensional box collection in \mathbf{R}^n , and let $\mathcal{X} = \{X_1, \dots, X_k\}$ be a finite set of pairwise disjoint semialgebraic sets in \mathbf{R}^n . We now define when \mathcal{B} and \mathcal{X} are in general position. We decompose $|\mathcal{B}|$ and \mathcal{X} into a finite number of regular sets and then define “being in general position” in terms of these decompositions as follows.

In (5.10), we defined a decomposition of a box collection into regular sets. Applied to $|\mathcal{B}|$, this results in the decomposition $|\mathcal{B}|_n, \dots, |\mathcal{B}|_0$, where $|\mathcal{B}|_i$ is a union of interiors of i -dimensional boxes in \mathbf{R}^n .

For each X_i , let R_{i0}, \dots, R_{in_i} be a regular decomposition of X_i . We say that \mathcal{B} and \mathcal{X} are in general position if and only if $\{|\mathcal{B}|_n, \dots, |\mathcal{B}|_0\}$ and $\{R_{1,0}, \dots, R_{1,n_1}, \dots, R_{k,0}, \dots, R_{k,n_k}\}$ are in general position.

We now describe the construction of an n -dimensional special box collection (the properties of this box collection will become clear later on). The construction takes as input

- a bounded semialgebraic set A in \mathbf{R}^n ;
- a uniform cone radius collection U_0, \dots, U_m of $\text{cl}(A)$ (as defined in section 5.2); and
- a fixed n -dimensional box collection \mathcal{F} in \mathbf{R}^n , which is in general position with $\{U_0, \dots, U_m\}$.

The result of the construction will be

- a set of box collections $\mathcal{R} = \{\mathcal{R}_0, \dots, \mathcal{R}_m\}$ and
- a positive real number δ

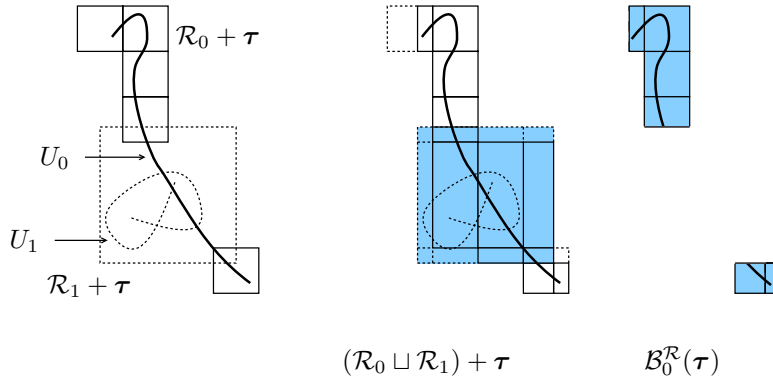


FIG. 6.1. Illustration of the construction of the box collection $\mathcal{B}_0^{\mathcal{R}}(\tau)$ for $\mathcal{R} = \{\mathcal{R}_0, \mathcal{R}_1\}$ and $U = U_0 \cup U_1$ as explained in Example 6.1. The picture shows $\mathcal{R} + \tau$ (right), the intermediate result $(\mathcal{R}_0 \sqcup \mathcal{R}_1) + \tau$ (middle), and the end result $\mathcal{B}_0^{\mathcal{R}}(\tau)$ (right).

satisfying some properties. Before we can state these properties, we need to define for $k = m, \dots, 0$ and $\tau \in \mathbf{R}^n$ the box collections

$$\mathcal{B}_k^{\mathcal{R}}(\tau) := (((\mathcal{R}_k \sqcup \dots \sqcup \mathcal{R}_m) + \tau \sqcup \mathcal{F}) \cap U_k) \setminus \{B' \in ((\mathcal{R}_k \sqcup \dots \sqcup \mathcal{R}_m) + \tau \sqcup \mathcal{F}) \cap U_k \mid |B'| \subseteq |\mathcal{B}_{k+1}^{\mathcal{R}}(\tau) \cup \dots \cup \mathcal{B}_m^{\mathcal{R}}(\tau)|\}.$$

In the following, we will write $\mathcal{B}_i^{\mathcal{R}}$ for $\mathcal{B}_i^{\mathcal{R}}(\mathbf{0})$ and let $U = U_0 \cup \dots \cup U_m$. The definition of $\mathcal{B}_k^{\mathcal{R}}(\tau)$ basically tells how to fit together all the box collections in \mathcal{R} and specifies which boxes should be disregarded. We illustrate the definition of $\mathcal{B}_k^{\mathcal{R}}$ by the following example.

Example 6.1. Assume we have a box collection $\mathcal{R} = \{\mathcal{R}_0, \mathcal{R}_1\}$ covering $U = U_0 \cup U_1$. In Figure 6.1 (left) we have depicted \mathcal{R}_0 and \mathcal{R}_1 with solid and dotted lines, respectively. Moreover, the set U_1 consists of the dotted curve, while U_0 is shown as a thick solid line. In this example, we assume that no fixed box collection \mathcal{F} is present.

Then by definition, $\mathcal{B}_1^{\mathcal{R}}(\tau) = (\mathcal{R}_1 + \tau) \cap U_1$. This box collection (in this example consisting of a single box only) corresponds to the large shaded box in Figure 6.1 (middle). For the construction of $\mathcal{B}_0^{\mathcal{R}}(\tau)$, we first compute the box collection $(\mathcal{R}_0 \sqcup \mathcal{R}_1) + \tau$, which consists of all the boxes shown in Figure 6.1 (middle). Solid-lined boxes intersect U_0 ; dotted-lined boxes do not. In order to obtain $\mathcal{B}_0^{\mathcal{R}}(\tau)$, all dotted-lined boxes are removed as well as those solid-lined boxes, which are included in $\mathcal{B}_1^{\mathcal{R}}(\tau)$ (the shaded area). The resulting box collection $\mathcal{B}_0^{\mathcal{R}}(\tau)$ is shown in Figure 6.1 (right). \diamond

We now continue with the statement of the desired properties of the box collection \mathcal{R} and real number δ . They must satisfy the properties

- (i) $\text{cl}(U)^\delta \subseteq \text{int}(|\mathcal{B}_0^{\mathcal{R}} \cup \dots \cup \mathcal{B}_m^{\mathcal{R}}|)$;
- (ii) for all $i = 0, \dots, m$ and for all $\tau \in \mathbf{R}^n$ of norm less than δ , $(\mathcal{R}_i + \tau) \sqcup \mathcal{F} \pitchfork U_i$; and
- (iii) for all $i = 0, \dots, m$ and for all $\tau \in \mathbf{R}^n$ of norm less than δ , and for each n -dimensional box $B \in \mathcal{B}_i^{\mathcal{R}}(\tau)$, there exists a point $\vec{p} \in \text{int}(|B| \cap U_i)$ such that $\gamma_{\text{cone}, A}(\vec{p}) > \text{diam}(B)$.

Construction algorithm. The construction of the box collection is done inductively on the number of parts m in the uniform cone radius collection $\{U_0, \dots, U_m\}$.

For the base case, when the uniform cone radius collection is empty, we define

$\mathcal{R}_{-1} = \emptyset$ and take $\delta = \infty$. Properties (i), (ii), and (iii) are then trivially satisfied.

Suppose now that U is nonempty and consists of m parts. By the induction hypothesis, there exist n -dimensional box collections $\mathcal{R}' = \{\mathcal{R}'_1, \dots, \mathcal{R}'_m\}$ and a positive real number δ' such that

- (i)' $\text{cl}(U \setminus U_0)^{\delta'} \subseteq \text{int}(|\mathcal{B}_1^{\mathcal{R}'} \cup \dots \cup \mathcal{B}_m^{\mathcal{R}'}|)$;
- (ii)' for all $i = 1, \dots, m$ and for all $\tau \in \mathbf{R}^n$ of norm less than δ' , $(\mathcal{R}'_i + \tau) \sqcup \mathcal{F} \pitchfork U_i$; and
- (iii)' for all $i = 1, \dots, m$ and for all $\tau \in \mathbf{R}^n$ of norm less than δ' , and for each n -dimensional box $B \in \mathcal{B}_i^{\mathcal{R}'(\tau)}$, there exists a point $\vec{p} \in \text{int}(|B| \cap U_i)$ such that $\gamma_{\text{cone}, A}(\vec{p}) > \text{diam}(B)$.

The construction consists of two steps:

Step 1. Cover the part of U_0 which may become uncovered by translations of the box collection $\mathcal{R}' + \tau$, for $\|\tau\| < \delta'$, with a box covering of a certain size. This size is determined by the uniform cone radius of the part of U_0 possibly uncovered by the translates of \mathcal{R}' .

Step 2. Some of the boxes in the above box covering might be in a degenerate position and in this way prevent the box collection from satisfying the required properties. This can be easily resolved, however, by translating all boxes with a small translation vector τ . Lemma 6.3 shows that it is possible to bring all boxes into general position; Lemma 6.4 shows that translating the boxes indeed results in a box collection with the desired properties.

We describe the two steps now in more detail. An example of the construction can be seen in Figure 6.2 and is described in the following example.

Example 6.2. We consider the case that no fixed box collection \mathcal{F} is present. Let $\{A_0, A_1\}$ be the uniform cone radius decomposition of $\text{cl}(A)$ (see Figure 6.2(a)). The set A_1 consists of the horizontal circle and point \vec{p} in Figure 6.2(a). The set A_0 is equal to the remainder $\text{cl}(A) \setminus A_1$.

1. *Base case* (not shown in Figure 6.2): $U = \emptyset, U_0 = \emptyset$. By definition, $\mathcal{R}_{-1} = \{\emptyset\}$, $\delta = \infty$.
2. *Case $m = 1, U = A_1, U_0 = A_1$.*

Covering U_0 : Since in Step 1 nothing is yet constructed, we have that $V = U_0, W = \emptyset$, and $\zeta = \infty$. Hence, $\mathcal{R}'' = \frac{\varepsilon V}{4\sqrt{3}}\text{-cover}(V)$. This box covering is depicted by the dashed boxes in Figure 6.2(a). By definition, $\delta'' = \min\{\frac{\delta'}{3} = \infty, \eta, \zeta = \infty\} = \eta$, where η is such that $\text{cl}(V)^\eta \subseteq \text{int}(|\mathcal{R}''|)$.

Translating \mathcal{R}'' : As can be seen in Figures 6.2(a), (b), the point \vec{p} lies on a side of one of the boxes at the bottom. In other words, \vec{p} is not in general position with the box collection. A simple small translation, however, resolves this situation and brings \vec{p} into general position with the box collection (see Figure 6.2(b)) while keeping the other points U_0 in general position as well. The resulting box collection is denoted by \mathcal{R} .

From \mathcal{R} we get $\mathcal{B}_0^{\mathcal{R}}$, as shown in Figure 6.2(c), by removing, in this case, a single box which no longer intersect U_0 .

3. *Case $m = 2, U = A_0 \cup A_1, U_0 = A_0, \mathcal{R}''_1 = \mathcal{R}$, and $\delta' = \delta$ (obtained in Step 2).*

Covering U_0 : We focus on a region around the box B in \mathcal{R}''_1 containing \vec{p} (See Figure 6.2(d)). For expository reasons, the position of U with respect to B is slightly simplified.

We have depicted the set V (dark shaded area) of points in U_0 , which might be outside $|B|$ when B is slightly translated, and show the remaining set W

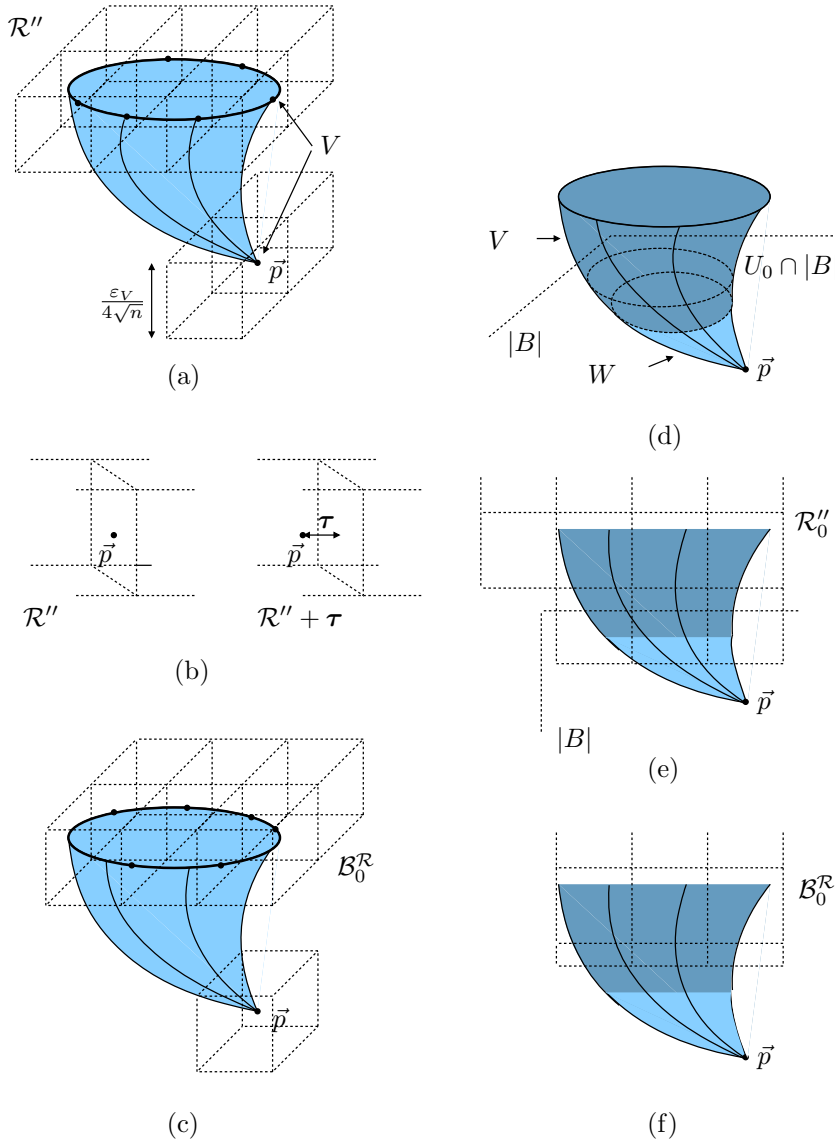


FIG. 6.2. Construction of the special box collection \mathcal{R} .

(light shaded area) as well. The new box collection \mathcal{R}_0'' will be $\frac{\varepsilon_V}{4\sqrt{3}}$ -cover(V). In order to not overload the figure, we have depicted the box collection from a sideways point of view (See Figure 6.2(e)). Let $\mathcal{R}'' = \{\mathcal{R}_0'', \mathcal{R}_1''\}$.

The constraint δ'' on the norm of translation vectors is given by $\delta'' = \min\{\frac{\delta'}{3}, \eta, \zeta\}$. It takes into account the distance between W and the boundary of the boxes constructed in Step 2 (ζ), the distance between V and the boundary of boxes in \mathcal{R}_0'' (η), and the constraint given in Step 2 (δ').

Translating \mathcal{R}'' : If necessary, slightly translate \mathcal{R}'' to bring it in general position such that it satisfies the desired properties. This results in the final box collection \mathcal{R} .

We also show part of $\mathcal{B}_0^{\mathcal{R}}$ (See Figure 6.2(f)). We refer to Example 6.1 for a discussion of its construction. The collection $\mathcal{B}_1^{\mathcal{R}}$ is equal to $\mathcal{B}_0^{\mathcal{R}}$ constructed in Step 2. \diamond

We now continue with the general description of the construction.

First step: Covering U_0 . We will define a set \mathcal{R}_0'' and define $\mathcal{R}_i'' = \mathcal{R}_i'$ for $i = 1, \dots, m$ such that for $\mathcal{R}'' = \{\mathcal{R}_0'', \dots, \mathcal{R}_m''\}$, $\text{cl}(U)\delta'' \subseteq \text{int}(|\mathcal{B}_0^{\mathcal{R}''}(\boldsymbol{\tau}) \cup \dots \cup \mathcal{B}_m^{\mathcal{R}''}(\boldsymbol{\tau})|)$ for some $\delta'' > 0$.

All points of U_0 that can become uncovered by varying the vector $\boldsymbol{\tau}$ in $|\mathcal{B}_1^{\mathcal{R}'}(\boldsymbol{\tau}) \cup \dots \cup \mathcal{B}_m^{\mathcal{R}'}(\boldsymbol{\tau})|$ with $\|\boldsymbol{\tau}\| < \frac{\delta'}{3}$ are included in the set

$$V := U_0 - (|\mathcal{B}_1^{\mathcal{R}'} \cup \dots \cup \mathcal{B}_m^{\mathcal{R}'}| - (\partial|\mathcal{B}_1^{\mathcal{R}'} \cup \dots \cup \mathcal{B}_m^{\mathcal{R}'}|)^{\frac{\delta'}{3}}).$$

By (i)', the minimal distance from any point in $U \setminus U_0$ to the boundary $\partial(|\mathcal{B}_1^{\mathcal{R}'} \cup \dots \cup \mathcal{B}_m^{\mathcal{R}'}|)$ is greater than or equal to δ' . This implies that

$$\text{cl}(U \setminus U_0)^{\frac{\delta'}{3}} \subseteq |\mathcal{B}_1^{\mathcal{R}'} \cup \dots \cup \mathcal{B}_m^{\mathcal{R}'}| - (\partial|\mathcal{B}_1^{\mathcal{R}'} \cup \dots \cup \mathcal{B}_m^{\mathcal{R}'}|)^{\frac{\delta'}{3}},$$

and hence, because U_0, \dots, U_m is a uniform cone radius collection, there exists a uniform cone radius, ε_V , of A for the set V . Let \mathcal{R}_0'' be $\frac{\varepsilon_V}{4\sqrt{n}}$ -cover(V). Note that

$$(6.1) \quad \text{diam}(B) = \frac{\varepsilon_V}{2}$$

for any box $B \in \mathcal{R}_0''$. The reason why we take this specific box covering is that the box collection, which we are constructing, must satisfy property (iii).

We now show that there exists a positive real number δ'' such that (i) holds for $\mathcal{R}'' = \{\mathcal{R}_0'', \dots, \mathcal{R}_m''\}$ and δ'' .

We partition $U_0 \cup \dots \cup U_m$ into three parts: $U \setminus U_0$, V , and

$$W := U_0 \cap (|\mathcal{B}_0^{\mathcal{R}''} \cup \dots \cup \mathcal{B}_m^{\mathcal{R}''}| - (\partial|\mathcal{B}_0^{\mathcal{R}''} \cup \dots \cup \mathcal{B}_m^{\mathcal{R}''}|)^{\frac{\delta'}{3}}).$$

By (i)',

$$(6.2) \quad \text{cl}(U \setminus U_0)^{\frac{\delta'}{3}} \subseteq \text{int}(|\mathcal{B}_1^{\mathcal{R}'} \cup \dots \cup \mathcal{B}_m^{\mathcal{R}'}|) \subseteq \text{int}(|\mathcal{B}_0^{\mathcal{R}''} \cup \dots \cup \mathcal{B}_m^{\mathcal{R}''}|).$$

We shall need the following lemma, which is readily verified.

LEMMA 6.1. *Let X and Y be two sets in \mathbf{R}^n . If X is bounded, then $\text{cl}(X) \subseteq \text{int}(Y)$ implies that there exists a positive real number ε such that $\text{cl}(X)^\varepsilon \subseteq \text{int}(Y)$.*

By the definition of a box covering, $\text{cl}(V) \subseteq \text{int}(|\mathcal{B}_0^{\mathcal{R}''}|) \subseteq \text{int}(|\mathcal{B}_0^{\mathcal{R}''} \cup \dots \cup \mathcal{B}_m^{\mathcal{R}''}|)$. Since A is bounded, V is also bounded. By Lemma 6.1, there exists a positive real number η such that

$$(6.3) \quad \text{cl}(V)^\eta \subseteq \text{int}(|\mathcal{B}_0^{\mathcal{R}''} \cup \dots \cup \mathcal{B}_m^{\mathcal{R}''}|).$$

We now prove that Lemma 6.1 can also be used for W .

LEMMA 6.2. $\text{cl}(W) \subseteq \text{int}(|\mathcal{B}_1^{\mathcal{R}'} \cup \dots \cup \mathcal{B}_m^{\mathcal{R}'}|)$.

Proof of Lemma 6.2. Suppose that there exists a point $\vec{p} \in \text{cl}(W)$ such that $\vec{p} \notin \text{int}(|\mathcal{B}_1^{\mathcal{R}'} \cup \dots \cup \mathcal{B}_m^{\mathcal{R}'}|)$. Let (\vec{p}_m) for $m > 0$ be a sequence of points in W such that $\|\vec{p} - \vec{p}_m\| < 1/m$. By the definition of W , for all points in $\vec{r} \in \partial|\mathcal{B}_1^{\mathcal{R}'} \cup \dots \cup \mathcal{B}_m^{\mathcal{R}'}|$, $\|\vec{r} - \vec{p}_m\| \geq \frac{\delta'}{3}$ for every m .

Now, every line segment $\{\lambda\vec{p}_m + (1-\lambda)\vec{p} \mid 0 \leq \lambda \leq 1\}$, intersects $\partial|\mathcal{B}_1^{\mathcal{R}'} \cup \dots \cup \mathcal{B}_m^{\mathcal{R}'}|$ in a point \vec{r}_m . However, since $\|\vec{p}_m - \vec{p}\| < 1/m$, also $\|\vec{p}_m - \vec{r}_m\| < 1/m$. Thus, we obtain a contradiction for m large enough such that $\frac{1}{m} < \frac{\delta'}{3}$. \square

Hence, by Lemma 6.1 and Lemma 6.2 there exists a positive real number ζ such that

$$(6.4) \quad W^\zeta \subseteq \text{int}(|\mathcal{B}_1^{\mathcal{R}'} \cup \dots \cup \mathcal{B}_m^{\mathcal{R}'}|) \subseteq \text{int}(|\mathcal{B}_0^{\mathcal{R}''} \cup \dots \cup \mathcal{B}_m^{\mathcal{R}''}|).$$

From the inclusions (6.2), (6.3), and (6.4), it follows that property (i) is satisfied for \mathcal{R}'' and δ'' , with $\delta'' = \min\{\frac{\delta'}{3}, \eta, \zeta\}$.

Second step: translating \mathcal{R}'' . The box collections in \mathcal{R}'' already satisfy property (i) for δ'' . However, properties (ii) and (iii) are not necessarily satisfied. This can be seen in Figure 6.2 (a), (b). We now show that a little translation of the box collection is all that is needed so that all properties are satisfied by the translated box collections.

LEMMA 6.3. *For each $i = 0, \dots, m$, there exists a translation $\tau \in \mathbf{R}^n$ of norm $\|\tau\| < \delta''$ such that*

$$(\mathcal{R}''_i + \tau) \sqcup \mathcal{F} \pitchfork U_i.$$

Proof of Lemma 6.3. Consider the decomposition of $|(\mathcal{R}''_i + \tau) \sqcup \mathcal{F}|$ into the sets $|(\mathcal{R}''_i + \tau) \sqcup \mathcal{F}|_j$ for $i = 0, \dots, m$ and for $j = 0, \dots, n$. Recall from section 5.5 that $|(\mathcal{R}''_i + \tau) \sqcup \mathcal{F}|_j$ is the union of the geometric realizations of boxes in $((\mathcal{R}''_0 + \tau) \sqcup \mathcal{F})^j$.

We need to prove that there exists a translation $\tau \in \mathbf{R}^n$, $\|\tau\| < \delta''$, such that for each $i = 0, \dots, m$, for each $r \in \{0, \dots, n_i\}$, for each $j \in \{0, \dots, n\}$, and for each $B \in ((\mathcal{R}''_i + \tau) \sqcup \mathcal{F})^j$, we have that

$$(6.5) \quad |B| \pitchfork R_{i,r}.$$

Let T denote the set of all possible translations: $T := \{\tau \in \mathbf{R}^n \mid \|\tau\| < \delta''\}$. Note that case $i > 0$ of (6.5) holds for any $\tau \in T$ by induction. Hence, we can focus on the case $i = 0$. Take an arbitrary B as in (6.5), take r arbitrary in $\{0, \dots, n\}$, and consider a point $\vec{x} \in |B| \cap R_{0,r}$. We are going to impose several conditions on T such that if $\tau \in T$ and τ satisfies these conditions, then (6.5) holds for τ . By definition of the union operator \sqcup , there exists a neighborhood W of \vec{x} such that one of the following three cases holds:

1. $|B| \cap W = |B'| \cap W$ for some $B' \in \mathcal{F}^p$ for some p . Note that

$$(6.6) \quad T_{\vec{x}}|B| = T_{\vec{x}}(|B| \cap W) = T_{\vec{x}}(|B'| \cap W) = T_{\vec{x}}|B'|.$$

Given that $\mathcal{F} \pitchfork U_0$, $|B'|$ and $R_{0,r}$ are transversal in \vec{x} for all $\tau \in T$. By (6.6), we may conclude that $|B|$ and $R_{0,r}$ are transversal in \vec{x} for all $\tau \in T$.

2. $|B| \cap W = |B''| \cap W$ for some $B'' \in (\mathcal{R}''_0 + \tau)^q$ for some q . Note that

$$(6.7) \quad T_{\vec{x}}|B| = T_{\vec{x}}(|B| \cap W) = T_{\vec{x}}(|B''| \cap W) = T_{\vec{x}}|B''|.$$

Suppose that

$$(T1) \quad (\mathcal{R}''_0 + \tau) \pitchfork U_0.$$

Then, $|B''| \pitchfork U_0$ and hence, $|B''|$ and $R_{0,r}$ are transversal in \vec{x} for all $\tau \in T$ such that condition (T1) is satisfied. By (6.7), we may conclude that $|B|$ and $R_{0,r}$ are transversal in \vec{x} for all $\tau \in T$ such that condition (T1) is satisfied.

3. $|B| \cap W = |B'| \cap |B''| \cap W$ for some $B' \in \mathcal{F}^p$ for some p , and for some $B'' \in (\mathcal{R}''_0 + \tau)^q$ for some q . Suppose that

$$(T2) \quad (\mathcal{R}''_0 + \tau) \pitchfork \mathcal{F}.$$

Because the intersection of regular sets in general position is regular, the tangent space $T_{\vec{x}}(|B'| \cap |B''|)$ exists. Note that

$$(6.8) \quad T_{\vec{x}}|B| = T_{\vec{x}}(|B| \cap W) = T_{\vec{x}}(|B'| \cap |B''| \cap W) = T_{\vec{x}}(|B'| \cap |B''|).$$

Furthermore, suppose that

$$(T3) \quad |B''| \pitchfork (|B'| \cap R_{0,r}).$$

When two regular sets intersect transversally in a point, the tangent space of the intersection in this point is the intersection of the tangent spaces of the regular sets in this point [23]. Hence, by (T2) and given that $\mathcal{F} \pitchfork U_0$, we have that $T_{\bar{x}}|B'| \cap T_{\bar{x}}|B''| = T_{\bar{x}}(|B'| \cap |B''|)$ and $T_{\bar{x}}|B'| \cap T_{\bar{x}}(R_{0,r}) = T_{\bar{x}}(|B'| \cap R_{0,r})$. Moreover, $T_{\bar{x}}(|B'| \cap R_{0,r}) \subseteq T_{\bar{x}}(R_{0,r})$. By (T3) we have that $T_{\bar{x}}(|B'| \cap |B''|) + T_{\bar{x}}(|B'| \cap R_{0,r}) = T_{\bar{x}}|B'|$. Hence,

$$\begin{aligned} T_{\bar{x}}|B| + T_{\bar{x}}(R_{0,r}) &= T_{\bar{x}}(|B'| \cap |B''|) + T_{\bar{x}}(R_{0,r}) \\ &= T_{\bar{x}}(|B'| \cap |B''|) + T_{\bar{x}}(|B'| \cap R_{0,r}) + T_{\bar{x}}(R_{0,r}) \\ &= T_{\bar{x}}(|B'|) + T_{\bar{x}}(R_{0,r}) \\ &= \mathbf{R}^n. \end{aligned}$$

Hence, we may conclude that $|B|$ and $R_{0,r}$ are transversal in \bar{x} for all $\tau \in T$ such that conditions (T2) and (T3) are satisfied.

We may conclude that $(\mathcal{R}_0'' + \tau) \sqcup \mathcal{F} \pitchfork U_0$ if $\tau \in T$ and if τ is such that, for each box $B \in ((\mathcal{R}_0'' + \tau) \sqcup \mathcal{F})^j$ for $j = 0, \dots, n$, either no extra condition holds, the condition (T1) holds, or both conditions (T2) and (T3) hold. Hence, we obtain a finite number of conditions on the translations in T . By Corollary 5.10, the set of translations $\tau \in T$ for which a single transversality condition, like (T1), (T2), or (T3), is not satisfied, has measure zero. Since a finite union of sets of measure zero also has measure zero, this implies that for almost all translations in T , all conditions can be satisfied simultaneously. This concludes the proof of the lemma. \square

Let τ_0 be a translation, as specified in Lemma 6.3. We now define for $i = 0, \dots, m$, $\mathcal{R}_i = \mathcal{R}_i'' + \tau_0$ and consider $\mathcal{R} = \{\mathcal{R}_0, \dots, \mathcal{R}_m\}$ and $\delta''' < \delta'' - \|\tau_0\|$.

LEMMA 6.4. *There exists a $\delta > 0$ such that $\mathcal{R}_0, \dots, \mathcal{R}_m$ and δ satisfy properties (i), (ii), and (iii).*

Proof of Lemma 6.4. We first prove that there exists a $\delta > 0$ such that property (ii) is satisfied. Indeed, the proof of Lemma 6.3 shows that for $i = 0, \dots, m$, $(\mathcal{R}_i'' + \tau) \sqcup \mathcal{F} \pitchfork U_i$ holds for any τ which satisfies a finite number of transversality conditions. Recall from section 5.4 that transversality is a stable property. Hence, if τ is a translation vector satisfying these transversality conditions, then there exists an $\varepsilon > 0$ such that any $\tau' \in \mathbf{R}^n$, for which $\|\tau' - \tau\| < \varepsilon$, also satisfies these transversality conditions.

Since $\mathcal{R}_i = \mathcal{R}_i'' + \tau_0$, and τ_0 is such that Lemma 6.3 holds, there exists a $\varepsilon > 0$ such that for $\tau \in \mathbf{R}^n$, $\|\tau\| < \varepsilon$,

$$(\mathcal{R}_i + \tau) \sqcup \mathcal{F} \pitchfork U_i$$

for $i = 0, \dots, m$. Hence, property (ii) is satisfied for $\mathcal{R}_0, \dots, \mathcal{R}_m$ and $\delta = \min\{\delta''', \varepsilon\}$.

We now prove that $\mathcal{R}_0, \dots, \mathcal{R}_m$ and δ also satisfy property (i). We will need the following properties which can be readily verified. Let X and Y be semialgebraic sets in \mathbf{R}^n . Then

- (1) $X^\varepsilon \subseteq Y \Rightarrow X \subseteq Y + \tau$ for any $\tau \in \mathbf{R}^n$ such that $\|\tau\| < \varepsilon$, and
- (2) $(X^{\varepsilon_1})^{\varepsilon_2} = X^{\varepsilon_1 + \varepsilon_2}$.

We already know $\text{cl}(U)^{\delta''} \subseteq \text{int}(|\mathcal{B}_0^{\mathcal{R}''} \cup \dots \cup \mathcal{B}_m^{\mathcal{R}''}|)$. Let $\varepsilon = \delta'' - \|\tau_0\| - \delta$. Since $\delta < \delta'' - \|\tau_0\|$, we have $\varepsilon > 0$, and by property (2),

$$\text{cl}(U)^{\delta''} = (\text{cl}(U)^\delta)^{\|\tau_0\| + (\delta'' - \|\tau_0\| - \delta)} \subseteq \text{int}(|\mathcal{B}_0^{\mathcal{R}''} \cup \dots \cup \mathcal{B}_m^{\mathcal{R}''}|).$$

By property (1), we have that

$$\text{cl}(U)^\delta \subseteq \text{int}(|\mathcal{B}_0^{\mathcal{R}''} \cup \dots \cup \mathcal{B}_m^{\mathcal{R}''}|) + \tau \quad \forall \tau : \|\tau\| < \|\tau_0\| + \varepsilon.$$

In particular, $\text{cl}(U)^\delta \subseteq \text{int}(|\mathcal{B}_0^{\mathcal{R}''} \cup \dots \cup \mathcal{B}_m^{\mathcal{R}''}|) + \tau_0 = \text{int}(|\mathcal{B}_0^{\mathcal{R}} \cup \dots \cup \mathcal{B}_m^{\mathcal{R}}|)$, and property (i) is satisfied for \mathcal{R} and δ .

We now prove that property (iii) is satisfied. Let $B \in \mathcal{B}_i^{\mathcal{R}}(\tau)$ for any $\tau \in \mathbf{R}^n$, $\|\tau\| < \delta$. We distinguish between the following two cases:

1. $i > 0$. Since $\mathcal{B}_i^{\mathcal{R}}(\tau) \subseteq \mathcal{B}_i^{\mathcal{R}'}(\tau_0 + \tau)$ and $\|\tau - \tau_0\| < \delta'$, we have by induction that there exists a $\vec{p} \in \text{int}(|B|) \cap U_i$ such that $\gamma_{\text{cone},A}(\vec{p}) > \text{diam}(B)$.
2. $i = 0$. Since $|B| \cap U_0 \neq \emptyset$, we need to prove that there exists a $\vec{p} \in \text{int}(|B|) \cap U_0$ such that $\gamma_{\text{cone},A}(\vec{p}) > \text{diam}(B)$.

Thus, let $\vec{x} \in |B| \cap U_0$. If $\vec{x} \in \text{int}(|B|)$, we are done. If $\vec{x} \in \partial|B|$, then $\vec{x} \in |B'| \cap U_0$ for some $|B'| \in ((\mathcal{R}_0 \sqcup \dots \sqcup \mathcal{R}_m) + \tau) \sqcup \mathcal{F}^p$ and some p . Let $D = (x_1 - \varepsilon, x_1 + \varepsilon, \dots, x_n - \varepsilon, x_n + \varepsilon)$ be an n -dimensional box centered around \vec{x} , with $\varepsilon \in \mathbf{R}$. For ε sufficiently small, $|B'| \cap \text{int}(|D|)$ has the form

$$(x_1 - \varepsilon, x_1 + \varepsilon) \times \dots \times (x_p - \varepsilon, x_p + \varepsilon) \times \{x_{p+1}\} \times \dots \times \{x_n\},$$

or a permutation of this form, which is handled analogously. Hence, $\text{int}(|B|) \cap \text{int}(|D|)$ has the form

$$(x_1 - \varepsilon, x_1 + \varepsilon) \times \dots \times (x_p - \varepsilon, x_p + \varepsilon) \times (x_{p+1}, x_{p+1} + \varepsilon) \times \dots \times (x_n, x_n + \varepsilon),$$

or a permutation of this form, which is handled analogously, or even a variant of this form, where some of the $n - p$ intervals $(x_i, x_i + \varepsilon)$ are replaced by $(x_i - \varepsilon, x_i)$, which again is handled analogously.

By property (ii),

$$(6.9) \quad \mathbb{T}_{\vec{x}}|B'| + \mathbb{T}_{\vec{x}}U_0 = \mathbf{R}^n.$$

Now, any $\vec{v} \in \mathbb{T}_{\vec{x}}|B'|$ is of the form $\vec{v} = (v_1, \dots, v_p, x_{p+1}, \dots, x_n)$; hence, by (6.9) there exists a tangent vector $\vec{w} \in \mathbb{T}_{\vec{x}}U_0$ such that $x_{p+1} < w_{p+1}, \dots, x_n < w_n$. By definition of the tangent space, if $\|\vec{w} - \vec{x}\|$ is small enough, there exists a point \vec{q} in U_0 arbitrarily close to \vec{w} . This point \vec{q} is also arbitrarily close to \vec{x} and also has $n - p$ last coordinates, which are strictly greater than the $n - p$ last coordinates of \vec{x} . Hence, \vec{q} is in $\text{int}(|B|) \cap \text{int}(|D|)$, and we have found a point in $\text{int}(|B|) \cap U_0$.

We now show that for any $\vec{p} \in \text{int}(|B|) \cap U_0$, $\gamma_{\text{cone},A}(\vec{p}) > \text{diam}(B)$. Indeed, any box in $\mathcal{B}_0^{\mathcal{R}}(\tau)$ is included in a box in $\mathcal{R}_0'' + \tau_0 + \tau$. By (6.1), \mathcal{R}_0'' consists of boxes which have a diameter that is strictly smaller than the uniform cone radius of $\text{int}(|B|) \cap U_0$. Hence, $\gamma_{\text{cone},A}(\vec{p}) > \text{diam}(B)$ for any point $\vec{p} \in \text{int}(|B|) \cap U_0$.

As a result, property (iii) is satisfied for \mathcal{R} and δ . □

This concludes the construction of the box collection \mathcal{R} and $\delta > 0$.

6.2. A first glance at the linearization algorithm. In this section we describe how the special box collection \mathcal{R} , constructed in the previous section, helps us in achieving our goal of linearizing a semialgebraic set $A \subseteq \mathbf{R}^n$.

First, using the box collection \mathcal{R} , we define

$$(6.10) \quad \mathcal{U} = \mathcal{B}_0^{\mathcal{R}} \cup \dots \cup \mathcal{B}_n^{\mathcal{R}}.$$

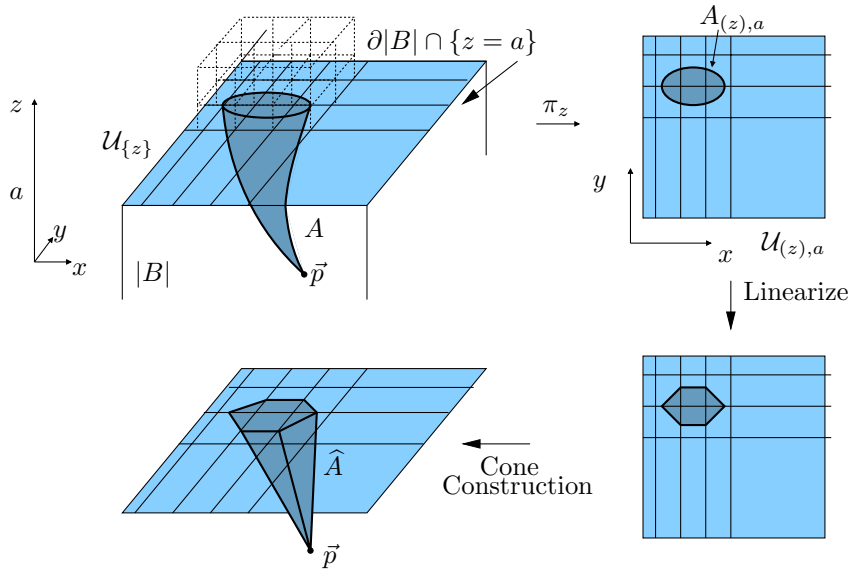


FIG. 6.3. Illustration of the linearization \widehat{A} inside $|B|$. The top side of $\partial|B|$ is shown together with that part of $\mathcal{U}_{(z)}$ and A lying on it. The top side has z -coordinate a (top left). The two-dimensional projected sets $\mathcal{U}_{(z),a}$ and $A_{(z),a}$ are shown (top right). The linearization algorithm is called inductively on these lower-dimensional sets (bottom right). The three-dimensional linearization consists of building a cone with top \vec{p} and base the previously constructed linearization on the boundary of B (bottom left).

Recall that $\mathcal{B}_i^{\mathcal{R}}$ stands for $\mathcal{B}_i^{\mathcal{R}}(\mathbf{0})$. Since each $\mathcal{B}_i^{\mathcal{R}}$ is a box collection and $\text{int}(|\mathcal{B}_i^{\mathcal{R}}|) \cap \text{int}(|\mathcal{B}_j^{\mathcal{R}}|) = \emptyset$ for any $i \neq j$, \mathcal{U} is a box collection too. It is clear that \mathcal{U} inherits some of the properties of \mathcal{R} . Indeed, by property (i) of \mathcal{R} , we know that \mathcal{U} is a box covering of $\text{cl}(A)$ and, by property (iii) of \mathcal{R} , we know that for each box $B \in \mathcal{U}$ there exists a point $\vec{p} \in \text{int}(|B|) \cap A$ such that $\gamma_{\text{cone},A}(\vec{p}) > \text{diam}(B)$.

The linearization algorithm, which will be described in more detail in section 6.3, works inductively on the boundaries of the boxes in \mathcal{U} . For each box $B \in \mathcal{U}$, the linearization algorithm replaces $|B| \cap A$ with a semilinear set in two steps. In the induction step, it replaces the intersection $\partial|B| \cap A$ with a semilinear set $\widehat{\partial|B| \cap A}$ on $\partial|B|$, which is homeomorphic to $\partial|B| \cap A$. Then, for each box $B \in \mathcal{U}$, it replaces $|B| \cap A$ with the semilinear set

$$\text{Cone}(\widehat{\partial|B| \cap A}, \vec{p}),$$

where $\vec{p} \in \text{int}(|B|) \cap A$ such that $\gamma_{\text{cone},A}(\vec{p}) > \text{diam}(B)$. It is shown in Lemma 6.5 that in this way we end up with a linearization of A . An illustration of the linearization algorithm is given in Figure 6.3.

In order to construct the linearization $\widehat{\partial|B| \cap A}$ on $\partial|B|$ of boxes $B \in \mathcal{U}$, we will need to construct again a box collection \mathcal{R} , but this time on the boundaries of the boxes in \mathcal{U} .

We will decompose the boundaries of the boxes in \mathcal{U} according to the direction of their supporting hyperplanes and according to the coordinate value of the fixed coordinate of these hyperplanes.

These coordinates can be computed as

$$\text{Coord}(\mathcal{U}_{\{i\}}) = \{a \in \mathbf{R} \mid \exists a_1, \exists b_1, \dots, \exists a_{i-1}, \exists b_{i-1}, \exists a_{i+1}, \exists b_{i+1}, \dots, \exists a_n, \exists b_n \\ (a_1, b_1, \dots, a_{i-1}, b_{i-1}, a, a, a_{i+1}, b_{i+1}, \dots, a_n, b_n) \in \mathcal{U}_{\{i\}}\}$$

for $i = 1, \dots, n$ and where $\mathcal{U}_{\{i\}}$ are the n -dimensional box collections defined in (5.9). Recall that $\mathcal{U}_{\{i\}}$ contains all n -dimensional boxes on the boundaries of boxes in \mathcal{U} , whose i th coordinates are all equal.

For each $a \in \text{Coord}(\mathcal{U}_{\{i\}})$, we will need all the points in $\text{cl}(A)$ with the i th coordinated fixed to a , i.e.,

$$\text{cl}(A)_{(i),a} := \{(x_1, \dots, x_{i-1}, x_{i+1}, \dots, x_n) \in \mathbf{R}^{n-1} \mid \\ (x_1, \dots, x_{i-1}, a, x_{i+1}, \dots, x_n) \in \text{cl}(A)\}$$

for $i = 1, \dots, n$.

Similarly, we define the $(n - 1)$ -dimensional box collections

$$\mathcal{U}_{(i),a} := \{(a_1, b_1, \dots, b_{i-1}, a_{i+1}, \dots, a_n, b_n) \in \mathbf{R}^{2(n-1)} \mid \\ (a_1, b_1, \dots, b_{i-1}, a, a, a_{i+1}, \dots, a_n, b_n) \in \mathcal{U}_{\{i\}}\}$$

for $i = 1, \dots, n$.

Since $\text{cl}(A) = C_0 \cup \dots \cup C_m$, and $C_m = R_{m,n} \cup \dots \cup R_{m,0}$, we have that

$$\text{cl}(A)_{(i),a} = (C_0)_{(i),a} \cup \dots \cup (C_m)_{(i),a}, \\ (C_j)_{(i),a} = (R_{j,n})_{(i),a} \cup \dots \cup (R_{j,0})_{(i),a}.$$

For each $i = 0, \dots, n$ and each $a \in \text{Coord}(\mathcal{U}_{\{i\}})$, we now show that we can construct an $(n - 1)$ -dimensional box collection \mathcal{R} , as described in section 6.1, for $\text{cl}(A)_{(i),a}$ in the role of $\text{cl}(A)$, $(C_0)_{(i),a}, \dots, (C_m)_{(i),a}$ in the role of, respectively, U_0, \dots, U_m , and $\mathcal{U}_{(i),a}$ in the role of \mathcal{F} .

However, for the construction to be successful, we need to verify that we start with valid input data. In other words, we need to show that $(C_0)_{(i),a}$ is a uniform cone radius with a regular decomposition given by $(R_{j,n})_{(i),a}$ and that \mathcal{F} (which is $\mathcal{U}_{(i),a}$) is in general position with $(C_0)_{(i),a}$ for the regular decomposition $(R_{j,n})_{(i),a}$.

CLAIM 6.1. *The sets $(C_0)_{(i),a}, \dots, (C_m)_{(i),a}$ form a uniform cone radius decomposition of $\text{cl}(A)_{(i),a}$.*

Proof. By definition, the sets $(C_0)_{(i),a}, \dots, (C_m)_{(i),a}$ form a decomposition of $\text{cl}(A)_{(i),a}$, so we need only show that each of the sets $(C_0)_{(i),a}, \dots, (C_m)_{(i),a}$ form a uniform cone radius collection.

We will need the following property, which is readily verified. Let X and Y be semialgebraic sets in \mathbf{R}^n . Then

- (1) if Y is closed and bounded, then for all ε' there exists an ε such that $X^\varepsilon \cap Y \subseteq (X \cap Y)^{\varepsilon'}$.

Let $H_{(i),a} = \{\vec{x} \in \mathbf{R}^n \mid x_i = a\}$, and let $\pi_i : \mathbf{R}^n \rightarrow \mathbf{R}^{n-1}$ be defined by $\pi_i(x_1, \dots, x_n) = (x_1, \dots, x_{i-1}, x_{i+1}, \dots, x_n)$. Let $j \in \{0, \dots, m\}$, and let $\varepsilon'_0, \dots, \varepsilon'_m$ be positive real numbers. We have that

$$(C_j)_{(i),a} \setminus \bigcup_{k=j+1}^m ((C_k)_{(i),a})^{\varepsilon'_k} = \pi_i \left((C_j \cap H_{(i),a}) \setminus \bigcup_{k=j+1}^m (C_k \cap H_{(i),a})^{\varepsilon'_k} \right).$$

By property (1), there exist $\varepsilon_0 > 0, \dots, \varepsilon_m > 0$ such that

$$(6.11) \quad (C_j \cap H_{(i),a}) \setminus \bigcup_{k=j+1}^m (C_k \cap H_{(i),a})^{\varepsilon'_k} \subseteq (C_j \cap H_{(i),a}) \setminus \bigcup_{k=j+1}^m (C_k^{\varepsilon_k} \cap H_{(i),a}) \\ = \left(C_j \setminus \bigcup_{k=j+1}^m C_k^{\varepsilon_k} \right) \cap H_{(i),a}.$$

Moreover, we have that $\text{cl}(A) = C_0 \cup \dots \cup C_m$ and, since C_0, \dots, C_m is a uniform cone radius collection, from the inclusion (6.11), it follows that

$$0 < \inf \left\{ \gamma_{\text{cone},A} \left(\left(C_j \setminus \bigcup_{k=j+1}^m C_k^{\varepsilon_k} \right) \cap H_{(i),a} \right) \right\} \\ \leq \inf \left\{ \gamma_{\text{cone},A} \left((C_j \cap H_{(i),a}) \setminus \bigcup_{k=j+1}^m (C_k \cap H_{(i),a})^{\varepsilon'_k} \right) \right\}.$$

We will next show that the following inequality holds:

$$\inf \left\{ \gamma_{\text{cone},A} \left((C_j \cap H_{(i),a}) \setminus \bigcup_{k=j+1}^m (C_k \cap H_{(i),a})^{\varepsilon'_k} \right) \right\} \\ \leq \inf \left\{ \gamma_{\text{cone},A \cap H_{(i),a}} \left((C_j \cap H_{(i),a}) \setminus \bigcup_{k=j+1}^m (C_k \cap H_{(i),a})^{\varepsilon'_k} \right) \right\} \\ = \inf \left\{ \gamma_{\text{cone},\pi_i(A \cap H_{(i),a})} \left(\pi_i \left((C_j \cap H_{(i),a}) \setminus \bigcup_{k=j+1}^m (C_k \cap H_{(i),a})^{\varepsilon'_k} \right) \right) \right\}.$$

Hence,

$$0 < \inf \left\{ \gamma_{\text{cone},A_{(i),a}} \left((C_j)_{(i),a} \setminus \bigcup_{k=j+1}^m ((C_k)_{(i),a})^{\varepsilon'_k} \right) \right\},$$

which proves that $(C_0)_{(i),a}, \dots, (C_m)_{(i),a}$ is a uniform cone radius collection.

We still need to prove that for each $\vec{x} \in C_j \cap H_{(i),a}$,

$$\gamma_{\text{cone},A}(\vec{x}) \leq \gamma_{\text{cone},A \cap H_{(i),a}}(\vec{x}).$$

The proof is illustrated in Figure 6.4. The main ingredient is the construction of the cone radius, as described in the proof of Theorem 2 in [14]. As explained in the paragraph immediately following Theorem 5.3, the radius query produces for each point \vec{x} an interval $(0, r)$ of cone radii, where r is the minimal distance between \vec{x} and each $\vec{s} \in \mathcal{S} \subseteq \mathbf{R}^n$, where \mathcal{S} contains those points \vec{s} which have a tangent space that is orthogonal to $\vec{x} - \vec{s}$ or parallel to one of the axes-parallel hyperplanes. Here, the tangent spaces are taken with respect to a Whitney-decomposition \mathcal{Z} of A , which is compatible with the union of all axes-parallel hyperplanes (including $H_{i,a}$) through \vec{x} . An example of such a Whitney-decomposition is given in Figure 6.4 (top right). Also in this figure, we have depicted the set \mathcal{S} . The (maximal) cone radius of A in (a, b) is illustrated by the dashed circle centered around (a, b) .

Recall that we defined

$$\gamma_{\text{cone},A}(\vec{x}) = \frac{1}{2}r = \frac{1}{2} \min_{\vec{s} \in \mathcal{S}} d(\vec{x}, \vec{s}),$$

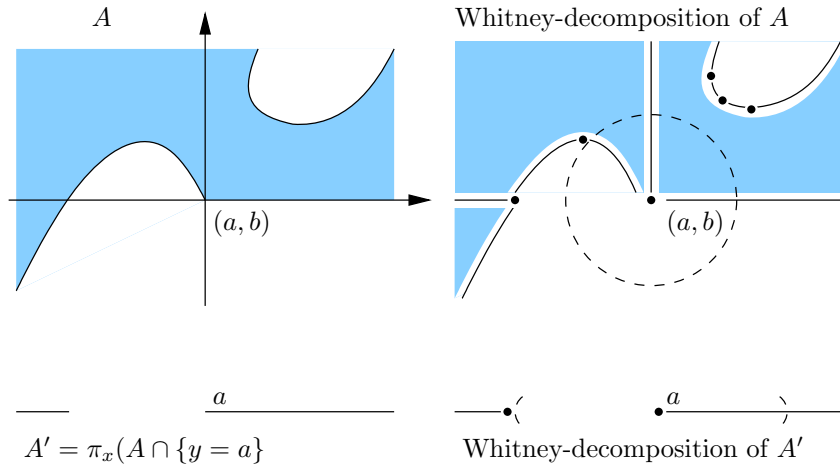


FIG. 6.4. Semialgebraic set A locally around (a, b) (top left). Whitney-decomposition \mathcal{Z} of A compatible with axes-parallel hyperplanes through (a, b) (top right). Intersection A' of A with horizontal hyperplane through (a, b) and projected on the x -axis (bottom left). Whitney-decomposition \mathcal{Z}' of A' (bottom right). The isolated points (top and bottom right) denote the critical points, i.e., points (c, d) with a horizontal or vertical tangent space, or a tangent space perpendicular to the vector $(c, d) - (a, b)$. Note that these tangent spaces are relative to the Whitney-decomposition. Moreover, by construction the set \mathcal{S} of critical points for A around (a, b) shown as the isolated points (top right) includes the set \mathcal{S}' of critical points of A' around a (bottom right). Consequently, $\gamma_{\text{cone}, A}(a, b) \leq \gamma_{\text{cone}, A'}(a)$.

where d denotes the ordinary distance function.

In the same way,

$$\gamma_{\text{cone}, A \cap H_{(i), a}}(\vec{x}) = \frac{1}{2} \min_{\vec{s} \in \mathcal{S}'} d(\vec{x}, \vec{s}),$$

where \mathcal{S}' contains those points \vec{s} which have a tangent space that is orthogonal to $\vec{x} - \vec{s}$ or parallel to one of the axes-parallel hyperplanes. Here, the tangent spaces are taken with respect to a Whitney-decomposition \mathcal{Z}' of $A \cap H_{(i), a}$. An example of such a Whitney-decomposition is given in Figure 6.4 (bottom right). Also in this figure we have depicted \mathcal{S}' . The (maximal) cone radius is illustrated by the interval bounded by the two dashed line segments centered around a .

Due to the requirement that \mathcal{Z} is compatible with the axes-parallel hyperplanes through \vec{x} , the Whitney-decomposition \mathcal{Z}' of $A \cap H_{(i), a}$ is equal to those strata $Z \in \mathcal{Z}$ such that $Z \subseteq H_{(i), a}$. In other words, $\mathcal{S}' \subseteq \mathcal{S}$, and hence,

$$\gamma_{\text{cone}, A}(\vec{x}) = \frac{1}{2} \min_{\vec{s} \in \mathcal{S}} d(\vec{x}, \vec{s}) \leq \frac{1}{2} \min_{\vec{s} \in \mathcal{S}'} d(\vec{x}, \vec{s}) = \gamma_{\text{cone}, A \cap H_{(i), a}}(\vec{x}),$$

as desired. \square

CLAIM 6.2. The sets $(R_{j,0})_{(i), a}, \dots, (R_{j,n_j})_{(i), a}$ form a regular decomposition of $(C_j)_{(i), a}$.

Proof. By definition, the sets $(R_{j,n})_{(i), a}, \dots, (R_{j,0})_{(i), a}$ form a decomposition of $(C_j)_{(i), a}$, so we need only show that each of the sets $(R_{j,k})_{(i), a}$, for $k = 0, \dots, n$, is regular. Let $H_{(i), a} = \{\vec{x} \in \mathbf{R}^n \mid x_i = a\}$, and let $\pi_i : \mathbf{R}^n \rightarrow \mathbf{R}^{n-1}$ be defined by $\pi_i(x_1, \dots, x_n) = (x_1, \dots, x_{i-1}, x_{i+1}, \dots, x_n)$.

It is sufficient to show that $R_{j,k}$ and $H_{(i), a}$ are in general position. Indeed, by the observation at the end of section 5.4, the intersection of two regular sets in general

position is again regular. Hence, $R_{j,k} \cap H_{(i),a}$ is regular. Thus, $(R_{j,k})_{(i),a} = \pi_i(R_{j,k} \cap H_{(i),a})$ is the image by the C^1 -diffeomorphism π_i of a regular set, and hence is regular itself.

We still need to show that $R_{j,k} \pitchfork H_{(i),a}$. By property (ii) of the constructed box collection \mathcal{U} , we know that $R_{j,k} \pitchfork \mathcal{U}$, and hence $R_{j,k} \pitchfork |\mathcal{U}|_\ell$. Let $\vec{x} \in R_{j,k} \cap H_{(i),a}$ and $B \in (\mathcal{U})^\ell$ such that $\vec{x} \in B \subset H_{(i),a}$. Note that such a B always exists because $a \in \text{Coord}(\mathcal{U}_{(i)})$ and \mathcal{U} covers A . Hence, $R_{j,k} \pitchfork |B|$ or, in other words, $T_{\vec{x}}R_{j,k} + T_{\vec{x}}|B| = \mathbf{R}^n$. Since $|B| \subset H_{(i),a}$, we have that $T_{\vec{x}}|B| \subseteq T_{\vec{x}}H_{(i),a}$, and hence also $T_{\vec{x}}R_{j,k} + T_{\vec{x}}H_{(i),a} = \mathbf{R}^n$. \square

CLAIM 6.3. *The box collections $\mathcal{U}_{(i),a}$ are in general position with $(C_0)_{(i),a}, \dots, (C_m)_{(i),a}$.*

Proof. We need to prove that $\{|\mathcal{U}_{(i),a}|_0, \dots, |\mathcal{U}_{(i),a}|_n\} \pitchfork \{(R_{j,k})_{(i),a} \mid j = 0, \dots, m, k = 0, \dots, n\}$. Let $H_{(i),a} = \{\vec{x} \in \mathbf{R}^n \mid x_i = a\}$, and let $\pi_i : \mathbf{R}^n \rightarrow \mathbf{R}^{n-1}$ be defined by $\pi_i(x_1, \dots, x_n) = (x_1, \dots, x_{i-1}, x_{i+1}, \dots, x_n)$.

We have that $|\mathcal{U}_{(i),a}|_\ell = \pi_i(|\mathcal{U}|_\ell \cap H_{(i),a})$. Thus, $B' \in (\mathcal{U}_{(i),a})^\ell$ if and only if $|B'| = \pi_i(|B|)$ with $B \in (\mathcal{U})^\ell$ and $|B| \subseteq H_{(i),a}$.

As already observed in the proof of Claim 6.2, $R_{j,k} \cap |\mathcal{U}|_\ell$ is a regular set. Hence, for $\vec{x} \in R_{j,k} \cap |\mathcal{U}|_\ell$ the tangent space $T_{\vec{x}}(R_{j,k} \cap |\mathcal{U}|_\ell)$ exists. Moreover, $T_{\vec{x}}(R_{j,k} \cap |\mathcal{U}|_\ell) = T_{\vec{x}}(R_{j,k} \cap |B|)$ for some $B \in (\mathcal{U})^\ell$ and $|B| \subseteq H_{(i),a}$.

Let $|B'| = \pi_i(|B|)$. We need to prove that

$$(6.12) \quad T_{\vec{x}_{(i),a}}|B'| + T_{\vec{x}_{(i),a}}((R_{j,k})_{(i),a}) = \mathbf{R}^{n-1}.$$

We have that

$$(6.13) \quad T_{\vec{x}_{(i),a}}|B'| = d\pi_i(T_{\vec{x}}|B|) \text{ and}$$

$$(6.14) \quad T_{\vec{x}_{(i),a}}((R_{j,k})_{(i),a}) = d\pi_i(T_{\vec{x}}(R_{j,k} \cap |B|)),$$

where $d\pi_i$ is the differential of π_i [23].

Moreover, because of property (ii) of the box collection \mathcal{U} and the remark at the end of section 5.4 on the intersection of tangent spaces, we have

$$(6.15) \quad T_{\vec{x}}|B| + T_{\vec{x}}(R_{j,k}) = \mathbf{R}^n \text{ and}$$

$$(6.16) \quad T_{\vec{x}}(R_{j,k} \cap |B|) = T_{\vec{x}}R_{j,k} \cap T_{\vec{x}}|B|.$$

Now, let $(v_1, \dots, v_{i-1}, v_{i+1}, \dots, v_n) \in \mathbf{R}^{n-1}$ and let $\vec{v} = (v_1, \dots, v_{i-1}, 0, v_{i+1}, \dots, v_n) \in \mathbf{R}^n$. By (6.15) there exists $\vec{b} \in T_{\vec{x}}|B|$ and $\vec{r} \in T_{\vec{x}}(R_{j,k})$ such that $\vec{v} = \vec{b} + \vec{r}$. Moreover, we may take $b_i = 0$ since vectors in $T_{\vec{x}}|B|$ have no component in the i th coordinate. Hence r_i has to be zero too. By (6.16), we have $\vec{r} \in T_{\vec{x}}(R_{j,k} \cap |B|)$. Let $\vec{b}' = d\pi_i(\vec{b})$ and $\vec{r}' = d\pi_i(\vec{r})$. Then by (6.13), $\vec{b}' \in T_{\vec{x}_{(i),a}}|B'|$, and by (6.14), $\vec{r}' \in T_{\vec{x}_{(i),a}}((R_{j,k})_{(i),a})$. By construction, $(v_1, \dots, v_{i-1}, v_{i+1}, \dots, v_n) = \vec{b}' + \vec{r}'$, proving (6.12). \square

6.3. Putting everything together: The linearization algorithm. The algorithm that constructs an \mathbf{A} -linear set, which is homeomorphic to a given semialgebraic set, works inductively on the dimension of the surrounding space in which the semialgebraic set is embedded.

6.3.1. The bounded case. The algorithm consists of two parts. The first part is a preprocessing step, which takes as input a bounded semialgebraic set A in \mathbf{R}^n and returns the regular decomposition of each part of the uniform cone radius decomposition of A .

Subroutine: PREPROCESS**Input:** A semialgebraic set A in \mathbf{R}^n .**Output:** A uniform cone radius decomposition C_0, \dots, C_k of A and for each C_i a regular decomposition $R_{i,0}, \dots, R_{i,i}$ of C_i .**Method:**

1. Compute the uniform cone radius decomposition of A :

$$A = C_0 \cup \dots \cup C_k.$$

2. Compute the regular decomposition of C_i , for $i = 0, \dots, k$:

$$C_i = R_{i,0} \cup \dots \cup R_{i,i}.$$

Subroutine: LINEARIZE-IN- n -DIMENSIONS**Input:** $(\{C_i\}, \{R_{i,r}\}, \mathcal{F})$ with C_0, \dots, C_k a uniform cone radius collection, $\{R_{i,r}\}$ a regular decomposition of C_i , and \mathcal{F} an n -dimensional box collection in \mathbf{R}^n which is in general position with C_0, \dots, C_k .**Output:** An \mathbf{A} -linear set \widehat{C} in \mathbf{R}^n which is homeomorphic to $C = C_0 \cup \dots \cup C_k$.**Method:**

- If $n > 1$, do the following:
 1. Compute the box collection \mathcal{U} constructed in section 6.2.
 2. Compute a $(3n+1)$ -ary relation \mathcal{P} consisting of pairs (B, \vec{p}_B, b) , where B is an n -dimensional box in \mathcal{U} , $\vec{p}_B \in \mathbf{R}^n$, and $b \in \{0, 1\}$ such that:
 - (a) $\vec{p}_B \in \text{cl}(C) \cap \text{int}(B)$ and is uniquely selected for each B ;
 - (b) $\gamma_{\text{cone}, C}(\vec{p}_B) > \text{diam}(B)$; and
 - (c) $b = 0$ in case $\vec{p}_B \in \text{cl}(C) \setminus C$ and $b = 1$ in case $\vec{p}_B \in C$.
 3. Compute all $\mathcal{U}_{(i),a}$ with $a \in \text{Coord}(\mathcal{U}_{\{i\}})$ and $i \in \{1, \dots, n\}$.
 4. Compute all $(C_j)_{(i),a} \subset \mathbf{R}^{n-1}$ with $a \in \text{Coord}(\mathcal{U}_{\{i\}})$ and $i \in \{1, \dots, n\}$.
 5. Compute all $(R_{i,r})_{(i),a} \subset \mathbf{R}^{n-1}$ with $a \in \text{Coord}(\mathcal{U}_{\{i\}})$ and $i \in \{1, \dots, n\}$.
 6. For all input triples $(\{(C_j)_{(i),a}\}, \{(R_{i,r})_{(i),a}\}, \mathcal{U}_{(i),a})$ with $a \in \text{Coord}(\mathcal{U}_{\{i\}})$ and $i \in \{1, \dots, n\}$, apply LINEARIZE-IN- $(n-1)$ -DIMENSIONS and embed the result in the corresponding hyperplane in \mathbf{R}^n , i.e., apply $(x_1, \dots, x_{n-1}) \mapsto (x_1, \dots, a, \dots, x_{n-1})$ where a appears in the i th position.
 7. Initialize \widehat{C} to the union of the results of the calls to LINEARIZE-IN- $(n-1)$ -DIMENSIONS of step 6.
- If $n = 1$, then do the following:
 1. Initialize \widehat{C} to $C_0 \cup \dots \cup C_k$.
- Output

$$\begin{aligned} \widehat{C} := & \widehat{C} \cup \{\text{Cone}(\widehat{C} \cap \partial B, \vec{p}_B) \mid (B, \vec{p}_B, b) \in \mathcal{P} \text{ and } b = 1\} \\ & \cup \{\text{Cone}(\widehat{C} \cap \partial B, \vec{p}_B) \setminus \{\vec{p}_B\} \mid (B, \vec{p}_B, b) \in \mathcal{P} \text{ and } b = 0\}. \end{aligned}$$

Algorithm: LINEARIZE

Input: A bounded semialgebraic set A in \mathbf{R}^n .

Output: An \mathbf{A} -linear set \widehat{A} in \mathbf{R}^n which is homeomorphic to A .

Method:

1. Call LINEARIZE-IN- n -DIMENSIONS(PREPROCESS(A), \emptyset).

Before we prove the correctness of the LINEARIZE algorithm, we want to point out the importance of the general position assumption made in the input of the algorithm. First of all, it allows us to treat all boxes in \mathcal{U} in the same way. More specifically, for every box B we are assured of having a point $\vec{p}_B \in \text{int}(|B|)$ as described in step 2 of the algorithm (see Lemma 6.4). The existence of these points is essential for the linearization, as is clear from the last step in the algorithm. Second, the general position assumption ensures that the lower-dimensional sets defined in steps 3–5 are nice and are again in general position (see the three claims in section 6.2). This implies that we can apply LINEARIZE on the lower-dimensional sets, which is a key feature for the algorithm.

LEMMA 6.5. *For any semialgebraic set A in \mathbf{R}^n , the set $\widehat{A} = \text{LINEARIZE}(A)$ is indeed a linearization of A .*

Proof. The linearity of \widehat{A} is immediate, so we focus on the existence of a homeomorphism $h : \mathbf{R}^n \rightarrow \mathbf{R}^n$, which maps A to \widehat{A} .

The existence proof (which is also a constructive proof) is an inductive proof. Before we can state the induction hypothesis, we need to define some box collections in \mathbf{R}^n .

We define $\mathcal{U}_{[n]}$ to be the n -dimensional box collection \mathcal{U} in \mathbf{R}^n constructed in step 1 when LINEARIZE-IN- n -DIMENSIONS is called.

Let $k < n$. With each call of LINEARIZE-IN- k -DIMENSIONS during the linearization of A we associate the pair $(i_{n-k}, a_{i-k}) \in \{1, \dots, n\} \times \mathbf{R}$ such that a_{n-k} is the value in $\text{Coord}(\mathcal{U}_{\{i_{n-k}\}})$ used in step 6. Note that \mathcal{U} is the box collection constructed in step 1 during the preceding call of LINEARIZE-IN- $(k + 1)$ -DIMENSIONS.

This sequence of pairs gives us a unique identifier for the box collection constructed in step 1 during each call of the algorithm. More specifically, we denote by $\mathcal{U}_{(i_1, a_1), \dots, (i_{n-k}, a_{n-k})}$ the box collection \mathcal{U} constructed in step 1 of the call LINEARIZE-IN- k -DIMENSIONS corresponding to (i_{n-k}, a_{n-k}) , which was called within LINEARIZE-IN- $(k + 1)$ -DIMENSIONS corresponding to (i_{n-k-1}, a_{n-k-1}) , and so forth until LINEARIZE-IN- $(n - 1)$ -DIMENSIONS is called with (i_1, a_1) within the initial call LINEARIZE-IN- n -DIMENSIONS. If $k = 1$, then no box collection \mathcal{U} is constructed, since step 1 is skipped in the algorithm. However, for the purpose of this proof, we define $\mathcal{U}_{(i_1, a_1), \dots, (i_{n-1}, a_{n-1})}$ to be $\mathcal{U}_{\{i_{n-1}\}, a_{n-1}}$, where \mathcal{U} is the box collection constructed in step 1 of the preceding call to LINEARIZE-IN-2-DIMENSIONS corresponding to (i_{n-2}, a_{n-2}) , and so forth.

At the same time, the sequence of pairs (i_j, a_j) tells us how to correctly embed $\mathcal{U}_{(i_1, a_1), \dots, (i_{n-k}, a_{n-k})}$ into \mathbf{R}^n . Indeed, the embedding simply maps $\vec{x} \in \mathbf{R}^k$ to the vector $\vec{x}' \in \mathbf{R}^n$ obtained by putting a_j at position i_j and filling up the k open slots with the values (in this order) x_1, \dots, x_k . We will denote this embedding by $\rho_{(i_1, a_1), \dots, (i_{n-k}, a_{n-k})}$.

We now define the k -dimensional box collection $\mathcal{U}_{[k]}$ in \mathbf{R}^n as

$$\mathcal{U}_{[k]} = \cup_{(i_1, a_1), \dots, (i_{n-k}, a_{n-k})} \rho_{(i_1, a_1), \dots, (i_{n-k}, a_{n-k})}(\mathcal{U}_{(i_1, a_1), \dots, (i_{n-k}, a_{n-k})}).$$

Let $\mathcal{U}_{[\leq k]}$ be the union of all boxes in $\mathcal{U}_{[k]}, \dots, \mathcal{U}_{[1]}$. We shall construct homeomorphisms $h_k : |\mathcal{U}_{[\leq k]}| \rightarrow |\widehat{A} \cap \mathcal{U}_{[\leq k]}|$, such that

- $h_k(A \cap |\mathcal{U}_{[\leq k]}|) = \widehat{A} \cap |\mathcal{U}_{[\leq k]}|$, and
- for all boxes B in $\mathcal{U}_{[k]}, \dots, \mathcal{U}_{[1]}$, $h_k|_{|B|} : |B| \rightarrow |B|$ is a homeomorphism.

We shall construct the homeomorphisms h_k by induction on k .

For the base case, $k = 1$, the linearization algorithm keeps A intact (see the case $n = 1$ in the description of the LINEARIZE-IN- n -DIMENSIONS algorithm). Hence, $\mathcal{U}_{[1]} \cap \widehat{A} = \mathcal{U}_{[1]} \cap A$ and we let h_k be the identity mapping on $\mathcal{U}_{[1]}$. Both conditions are trivially satisfied for h_1 .

Suppose we have constructed a homeomorphism $h_{k-1} : |\mathcal{U}_{[\leq k-1]}| \rightarrow |\widehat{A} \cap \mathcal{U}_{[\leq k-1]}|$ such that

- $h_{k-1}(A \cap |\mathcal{U}_{[\leq k-1]}|) = \widehat{A} \cap |\mathcal{U}_{[\leq k-1]}|$, and
- for all boxes B in $\mathcal{U}_{[k]}, \dots, \mathcal{U}_{[1]}$, $h_{k-1}|_{|B|} : |B| \rightarrow |B|$ is a homeomorphism.

Let $B' \in \mathcal{U}_{[k]}$; then we will define $h_k|_{|B'|} : |B'| \rightarrow |B'|$ as the composition of two homeomorphisms f and g . Let us first describe the homeomorphism g . By definition, $|B'| = \rho_{(i_1, a_1), \dots, (i_{n-k}, a_{n-k})}(|B|)$ with $B \in \mathcal{U}_{(i_1, a_1), \dots, (i_{n-k}, a_{n-k})}$.

Let \mathcal{P} be the relation computed in step 2 after $\mathcal{U}_{(i_1, a_1), \dots, (i_{n-k}, a_{n-k})}$ was computed. By the definition of the relation \mathcal{P} and by Theorem 5.2, there exists a homeomorphism $g|_{|B|} : |B| \rightarrow |B|$ such that $g|_{\partial|B|}$ is the identity, and either

1. $g|_{|B|}(|B| \cap A) = \text{Cone}(A \cap \partial|B|, \vec{p}_B)$ in case $(B, \vec{p}_B, 1) \in \mathcal{P}$, or
2. $g|_{|B|}(|B| \cap A) = \text{Cone}(A \cap \partial|B|, \vec{p}_B) \setminus \{\vec{p}_B\}$ in case $(B, \vec{p}_B, 0) \in \mathcal{P}$.

Since the second case is completely analogous to the first case, we assume that the first case holds for g . This concludes the description of the homeomorphism g .

Before we explain the construction of the second homeomorphism f , we show how to partition $|B|$ using the boundary of boxes $|B_t|$ parametrized by $t \in [0, 1]$. Suppose that $|B| = [a_1, b_1] \times \dots \times [a_n, b_n]$, and suppose $\vec{p}_B = (c_1, \dots, c_n)$ with $a_i < c_i < b_i$ for $i = 1, \dots, n$. Then the following sets for $0 \leq t \leq 1$ partition $|B|$ such that $|B| = \cup_{t \in [0, 1]} \partial|B_t|$:

$$|B_t| := [ta_1 + (1-t)c_1, tb_1 + (1-t)c_1] \times \dots \times [ta_n + (1-t)c_n, tb_n + (1-t)c_n] \quad 0 \leq t \leq 1.$$

Let $\vec{x} \in |B|$. To start with the construction of $f(\vec{x})$ for $\vec{x} \in |B|$, we define the unique t_0 such that $\vec{x} \in \partial|B_{t_0}|$. Then, let L be the halfline from \vec{p}_B to \vec{x} and define

$$\vec{y} = L \cap \partial|B|.$$

Next, let L' is the halfline from \vec{p}_B to $h_{k-1}(\vec{y})$. Note that $h_{k-1}(\vec{y})$ still lies on the boundary $\partial|B|$. Finally, define $f|_{|B|} : |B| \rightarrow |B|$ in \vec{x} as

$$f|_{|B|}(\vec{x}) = \partial|B_{t_0}| \cap L'.$$

The construction of f is illustrated in Figure 6.5. It can easily be verified that $f|_{|B|}$ is a homeomorphism from $|B|$ to $|B|$ such that

$$(6.17) \quad f|_{|B|}(\text{Cone}(A \cap \partial|B|, \vec{p}_B)) = \text{Cone}(h_{k-1}(A \cap \partial|B|), \vec{p}_B).$$

Finally, we define $h_k|_{|B'|} : |B'| \rightarrow |B'|$ using the composition of the two homeomorphisms $f|_{|B|}$ and $g|_{|B|}$, i.e.,

$$h_k|_{|B'|} = \rho_{(i_1, a_1), \dots, (i_{n-k}, a_{n-k})} \circ f|_{|B|} \circ g|_{|B|} \circ \pi_{i_1, \dots, i_{n-k}}.$$

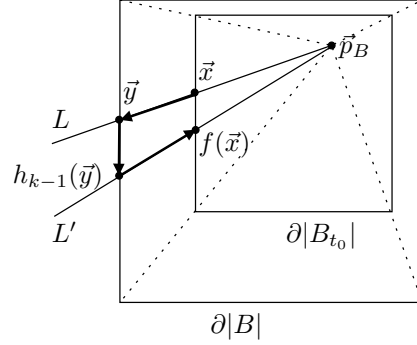


FIG. 6.5. Construction of the homeomorphism $f : |B| \rightarrow |B|$. The figure shows the construction of $f(\vec{x})$ for a point $\vec{x} \in |B|$.

We now define $h_k : |\mathcal{U}_{[\leq k]}| \rightarrow |\mathcal{U}_{[\leq k]}|$ as

$$h_k := \bigcup_{B \in \mathcal{U}_{[k]}} h_k|_{|B|}$$

and show that it has the desired properties. First, we prove that h_k is a homeomorphism. By the gluing Lemma [35, Lemma 3.8], it is sufficient to show that for any two boxes B and B' in $\mathcal{U}_{[k]}$, we have that

$$h_k|_{|B| \cup |B'|} = h_k|_{|B|} \cup h_k|_{|B'|} : |B| \cup |B'| \rightarrow |B| \cup |B'|.$$

For this to hold, it is sufficient to show that for any k -dimensional box $B' \in \mathcal{U}_{[k]}$ in \mathbf{R}^n ,

$$(6.18) \quad (h_k|_{|B|})|_{|B'|} = (h_k|_{|B'|})|_{|B|}.$$

This holds indeed. If $|B| \cap |B'| = \emptyset$, then we are done. Suppose that $\vec{x} \in |B| \cap |B'|$. Then by the definition of a box collection, $\vec{x} \in \partial|B| \cap \partial|B'|$. Now, for every box $B'' \in \mathcal{U}_{[k]}$, we have $h_k|_{\partial|B''|}(\vec{x}) = f|_{\partial|B''|}(\vec{x}) = h_{k-1}(\vec{x})$. Hence,

$$\begin{aligned} (h_k|_{|B|})|_{|B'|}(\vec{x}) &= h_k|_{\partial|B| \cap \partial|B'|}(\vec{x}) \\ &= h_{k-1}(\vec{x}) \\ &= h_k|_{\partial|B'| \cap \partial|B|}(\vec{x}) \\ &= (h_k|_{|B'|})|_{|B|}(\vec{x}). \end{aligned}$$

Hence, $h_k : |\mathcal{U}_{[\leq k]}| \rightarrow |\mathcal{U}_{[\leq k]}|$ is a homeomorphism.

Second, we show that for all boxes B in $\mathcal{U}_{[k]}, \dots, \mathcal{U}_{[1]}$, $h_{k-1}|_{|B|} : |B| \rightarrow |B|$ is a homeomorphism. By construction, this holds for any box $B \in \mathcal{U}_{[k]}$. For boxes $B' \in \mathcal{U}_{[i]}$ for $i < k$, it is sufficient to observe that such boxes B' lie on the boundary of a box B in $\mathcal{U}_{[k]}$, and on these boundaries h_k coincides with h_{k-1} for which the desired property holds by induction.

Finally, we still need to verify that $h_k(A \cap |\mathcal{U}_{[\leq k]}|) = \widehat{A} \cap |\mathcal{U}_{[\leq k]}|$. It is sufficient to show that $h_k(A \cap |B|) = \widehat{A} \cap |B|$ for any $B \in \mathcal{U}_{[k]}$. By (6.17), the induction hypothesis,

and the definition of \widehat{A} in the algorithm LINEARIZE-IN- n -DIMENSIONS, we have

$$\begin{aligned} h_k(A \cap |B|) &= \text{Cone}(h_{k-1}(A \cap \partial|B|), \vec{p}_B) \\ &= \text{Cone}(\widehat{A} \cap \partial|B|, \vec{p}_B) \\ &= \widehat{A} \cap |B|. \end{aligned}$$

Since $|\mathcal{U}|$ is closed, a standard result from topology [36] implies that the final homeomorphism h_n can be extended to a homeomorphism $h : \mathbf{R}^n \rightarrow \mathbf{R}^n$. \square

We are now ready to state the main result of this section.

THEOREM 6.6. *For each n , there exists an FO+POLY+TC formula `linearize` over the schema $\mathcal{S} = \{S\}$, with S an n -ary relation name such that for any polynomial constraint database D over \mathcal{S} , `linearize`(D) is an algebraic linearization of S^D if S^D is bounded.*

Proof. The desired FO+POLY+TC formula `linearize` expresses the algorithm LINEARIZE described above. From Lemma 5.5 and Lemma 5.7, it follows that the algorithm PREPROCESS is FO+POLY-expressible.

Concerning the algorithm LINEARIZE-IN- n -DIMENSIONS, we have that in step 1, the box collection \mathcal{U} is computed. In the construction of this box collection in section 6.1, we need to compute the following:

- The computation of a uniform cone radius. This is FO+POLY-expressible by Theorem 5.3.
- The computation of a finite number of box coverings, i.e., the $\frac{\varepsilon V}{4\sqrt{n}}$ -cover(V) coverings of section 6.1. This is FO+POLY+TC-expressible by Proposition 5.14.
- A candidate $\tau \in \mathbf{R}^n$ as specified in Lemma 6.3 needs to be found. Since this is essentially checking a finite number of transversality conditions, this is FO+POLY-expressible by Lemma 5.8.

Hence, we may conclude that the computation of \mathcal{U} is in FO+POLY+TC. In step 2, the relation \mathcal{P} is constructed. Given the box collection \mathcal{U} , we know by property (iii) of this collection that in each $B \in \mathcal{U}$ there exists a point $\vec{p} \in \text{int}(|B|) \cap \text{cl}(C)$ such that $\gamma_{\text{cone}, C}(\vec{p}) > \text{diam}(B)$. The set of points in $\text{int}(|B|)$ with this property is FO+POLY-expressible by Theorem 5.3. Hence, we can also select in FO+POLY for each $B \in \mathcal{U}$ a unique representant among these points. This will be \vec{p}_B . Hence, we may conclude that the computation of the relation \mathcal{P} is FO+POLY-expressible. In steps 3, 4, and 5, we need to compute $\text{Coord}(\mathcal{U}_{\{i\}})$, $\mathcal{U}_{(i), a}$, $(C_j)_{(i), a}$, and $(R_{i, r})_{(i), a}$. By definition these are all FO+POLY-expressible. In step 6 we call the algorithm n times. We have to be careful of how the inductive step is translated in FO+POLY+TC. A straightforward translation would result in a parametrized call of the transitive closure operators in the computation of the box coverings in step 1. Observe, however, that the set of parameters $\text{Coord}(\mathcal{U}_{(i)})$ for $i = \{1, \dots, n\}$ can be computed inside the transitive closure operator and that these parameters can then be passed on outside the transitive closure operator by simply annotating the vectors inside the transitive closure with these parameters. Indeed, suppose that we want to compute the transitive closure of a parametrized set $X \in \mathbf{R}^{n+m}$, where the last m coordinates are the parameters. Suppose that the set of parameters is FO+POLY+TC-definable from the database by a formula φ . We now define $Y = [\text{TC}_{\vec{x}, \vec{a}; \vec{y}, \vec{b}} X \wedge \vec{a} = \vec{b} \wedge \varphi(\vec{a})]$. We can then uniquely identify the result of this transitive closure computation for each parameter value by asking for all $(\vec{x}, \vec{a}) \in Y$, for which $\varphi(\vec{a})$ holds. By adapting the box covering formula constructed in Proposition 5.14, we can compute the box coverings for the parameter

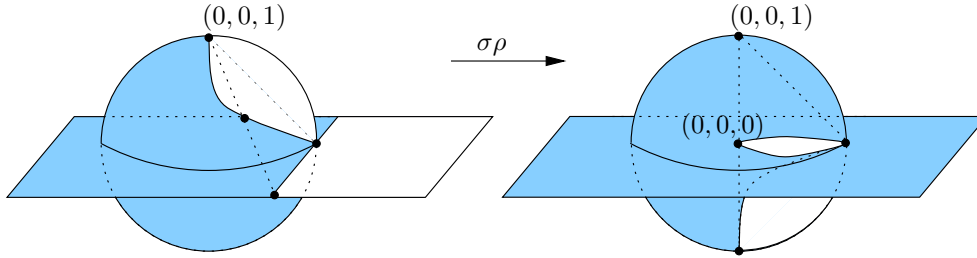


FIG. 6.6. A semialgebraic set (shaded area) is mapped onto the sphere $S^2(\vec{0}, 1)$, flipped vertically, and projected back onto the sphere $S^2(\vec{0}, 1)$. This brings the point at infinity \vec{p}_∞ to the origin $\vec{0}$.

set $\text{Coord}(\mathcal{U}_{(i)})$ in parallel and keep them apart afterwards. In this way, we do not need parametrized transitive closure, and hence step 6 is expressible in FO+POLY+TC.

In step 7 a simple union is performed (which is trivially in FO+POLY), and finally, the cones are constructed, which is also clearly expressible in FO+POLY.

Since the recursion depth is bounded by the dimension, we can write the complete execution of the algorithm as a single FO+POLY+TC formula. \square

If the linearization obtained in Theorem 6.6 also needs to be a good approximation from a metrical point of view, we can easily adapt the algorithms such that the approximation lies arbitrarily close to the original polynomial constraint database. Indeed, we can simply bound the diameter of the boxes used in the construction by a specified ε -value. We will see some applications of these ε -approximations in the next section.

THEOREM 6.7. *For each n there exists an FO+POLY+TC query ε -approx over the schema $\mathcal{S} = \{S\}$ with S an n -ary relation name such that for any polynomial constraint database D over \mathcal{S} such that S^D is bounded, the set ε -approx(D) is an algebraic ε -approximation of S^D .*

Proof. The proof follows at once from the fact that the homeomorphism h constructed in the proof of Theorem 6.6 maps $A \cap |B|$ to $\hat{A} \cap |B|$ for each box $B \in \mathcal{U}$. Thus, if $\vec{p} \in A \cap |B|$ then also $h(\vec{p}) \in |B|$. Because $\text{diam}(B) < \varepsilon$, the distance between \vec{p} and $h(\vec{p})$ is smaller than ε , so in this case \hat{A} will be an ε -approximation of A . \square

6.3.2. The general case. Let A be an unbounded semialgebraic set in \mathbf{R}^n . We reduce the construction of an algebraic linearization of A to the construction for bounded semialgebraic sets as follows.

First, we need to define the *cone radius of A in the point at infinity \vec{p}_∞* . Consider the embedding $i : \mathbf{R}^n \rightarrow \mathbf{R}^{n+1} : (x_1, \dots, x_n) \mapsto (x_1, \dots, x_n, 0)$. Let $\rho : \mathbf{R}^{n+1} \rightarrow \mathbf{R}^{n+1}$ be the reflection defined by $(x_1, \dots, x_{n+1}) \mapsto (x_1, \dots, x_n, -x_{n+1})$. Let $\mathbf{R}^n \cup \{\vec{p}_\infty\}$ be the one-point compactification of \mathbf{R}^n [35]. Finally, consider the stereographic projection $\sigma : S^n((0, \dots, 0), 1) \rightarrow i(\mathbf{R}^n) \cup \{\vec{p}_\infty\}$ defined by $\sigma(x_1, \dots, x_{n+1}) = \frac{(x_1, \dots, x_n)}{1-x_{n+1}}$ and $\sigma(0, \dots, 0, 1) = \vec{p}_\infty$.

We define a cone radius of A at \vec{p}_∞ as a cone radius of the semialgebraic set

$$i^{-1}(\sigma(\rho(\sigma^{-1}(i(A) \cup \{\vec{p}_\infty\}))))$$

in the origin of \mathbf{R}^n . Figure 6.6 illustrates the above transformation process. The local conic structure of semialgebraic sets implies that there exists an $m > 0$ such that $\{\vec{x} \in \mathbf{R}^n \mid \|\vec{x}\| \geq m\} \cap A$ is topologically equivalent to $\{\lambda \vec{x} \in \mathbf{R}^n \mid \vec{x} \in \partial([-m, m] \times \dots \times [-m, m]) \cap A \wedge \lambda \geq 1\}$.

We now present the unbounded version of the algorithm LINEARIZE.

Algorithm LINEARIZE'

Input: A semialgebraic set A in \mathbf{R}^n .

Output: An \mathbf{A} -linear set \widehat{A} in \mathbf{R}^n which is homeomorphic to A .

Method:

1. Compute a cone radius m of A in \vec{p}_∞ . Let $M = [-m, m] \times \cdots \times [-m, m]$.
2. Call $\text{Linearize}(A \cap M)$.
3. Output

$$\widehat{A} := (\widehat{A \cap M}) \cup \{\lambda \vec{x} \in \mathbf{R}^n \mid \vec{x} \in A \cap \partial M \wedge \lambda \geq 1\}.$$

We obtain the following generalization of Theorem 6.6.

THEOREM 6.8. *For each n there exists an FO+POLY+TC formula **linearize** over the schema $\mathcal{S} = \{S\}$, with S an n -ary relation name such that for any polynomial constraint database D over \mathcal{S} , $\text{linearize}(D)$ is an algebraic linearization of S^D .*

6.4. Rational linearizations. We now refine the previous theorems to *rational linearization*.

THEOREM 6.9. *For each n , there exists an FO+POLY+TC query **ratlin** over the schema $\mathcal{S} = \{S\}$, with S n -ary, such that for any polynomial constraint database D over \mathcal{S} such that S^D is bounded, $\text{ratlin}(D)$ is a rational linearization of S^D .*

Proof. We can obtain this result easily by modifying the construction of the special box collection in section 6.1 in the following way. When the box covering \mathcal{V} of size $\frac{\varepsilon \mathcal{V}}{\sqrt{n}}$ is computed in this construction, we compute a rational number that is smaller than $\frac{\varepsilon \mathcal{V}}{\sqrt{n}}$ and take this as the size of the box covering \mathcal{V} to be computed. By similar techniques as those in section 4, it is easy to show that there exists an FO+POLY+TC query, which returns a rational number smaller than the input number. In this way, all boxes in $\mathcal{R} \subset \mathbf{Q}^{2n}$. A second adaptation is that the relation \mathcal{P} is replaced by the following relation

$$\mathcal{P}' = \{(B, \vec{c}_B, b) \in \mathcal{U} \times \mathbf{Q}^n \times \{0, 1\} \mid \exists \vec{p}_B(B, \vec{p}_B, b) \in \mathcal{P}\},$$

where \vec{c}_B denote the center of the box B .

In this way the algorithm LINEARIZE-IN- n -DIMENSIONS will select points with rational coordinates. □

We also have a rational equivalent of Theorem 6.7.

THEOREM 6.10. *For each n there exists an FO+POLY+TC query ε -**ratlin** over the schema $\mathcal{S} = \{S\}$, with S an n -ary relation name, such that for any polynomial constraint database D over \mathcal{S} such that S^D is bounded, the set ε -**ratlin**(D) is a rational ε -approximation of S^D .*

6.5. The connectivity query. Although we know already that the connectivity query, which asks whether a polynomial constraint database is connected, is expressible in FO+POLY+TCS, we show in this section that the connectivity query is already expressible in FO+POLY+TC. Let A be a semialgebraic set in \mathbf{R}^n . For semialgebraic sets, expressing the connectivity query is the same as expressing whether any two points can be connected by a path lying entirely in A [6, Proposition 2.5.13]. One

can even choose the paths to be semialgebraic, in case of a semialgebraic set, and semilinear, in case of a semilinear set [44, Chapter 6, Proposition 3.2].

We now show that this query can be expressed in FO+POLY+TC using the formula `linearize` given in Theorem 6.8.

Let $\mathcal{S} = \{S\}$, with S an n -ary relation name. Consider the FO+POLY+TC formula `lineconn`(\vec{r}, \vec{s}) over \mathcal{S} such that for any database D over \mathcal{S} , $(\vec{p}, \vec{q}) \in \text{lineconn}(D)$ if and only if

$$\forall \lambda (0 \leq \lambda \leq 1), \quad \lambda \vec{p} + (1 - \lambda) \vec{q} \in \text{linearize}(D).$$

Define now the FO+POLY+TC sentence `connected`, which tests for any database D over \mathcal{S} whether

$$\forall \vec{p} \in \text{linearize}(D), \forall \vec{q} \in \text{linearize}(D), \quad (\vec{p}, \vec{q}) \in [\text{TC}_{\vec{x}; \vec{y}} \text{lineconn}(D)].$$

PROPOSITION 6.11. *Let $\mathcal{S} = \{S\}$ with S an n -ary relation name. The FO+POLY+TC formula `connected` always terminates and expresses the connectivity query.*

Proof. Since `linearize`(D) is topologically equivalent to S^D , S^D is connected if and only if `linearize`(D) is. Since `linearize`(D) is semilinear, two points \vec{p} and \vec{q} belong to the same connected component of `linearize`(D) if and only if there exists a piecewise linear path from \vec{p} to \vec{q} lying entirely in `linearize`(D). The formula `connected` expresses that all points of `linearize`(D) belong to the same connected component, i.e., that `linearize`(D) is connected.

To conclude that the evaluation of the transitive closure in the formula `connected` ends in finitely many steps, we need to show that there exists an upper bound on the number of line segments in `linearize`(D), which is needed to connect any two points in the same connected component of `linearize`(D). Now, any semilinear set can be decomposed into a finite number of convex sets [44]. The finiteness of this decomposition yields the desired bound. \square

Since FO+POLY+TC is included in stratified DATALOG with polynomial constraints, Proposition 6.11 solves the question [15, 31, 33] of whether stratified DATALOG with polynomial constraints can express the connectivity query.

6.6. Volume approximation. In this section, we shall use the box covering and the ε -approximation to approximate the volume of semialgebraic sets with an FO+POLY+TC formula. We restrict our attention to bounded semialgebraic sets and require that the evaluation of this FO+POLY+TC formula is effective for all bounded semialgebraic inputs.

Let $\mathcal{S} = \{S\}$, with S an n -ary relation name. Let D be a polynomial constraint database over \mathcal{S} .

The *volume* of a database D is defined as the Lebesgue-measure of the semialgebraic set $S^D \subseteq \mathbf{R}^n$ and is denoted by $\text{VOL}(D)$.

Since we want an FO+POLY+TC formula whose evaluation is effective on all databases, it is impossible to define the *exact* volume of polynomial constraint databases in FO+POLY+TC. Indeed, consider the database consisting of the unit disk D in \mathbf{R}^2 . The volume of D equals π . Since π is not algebraic, this value cannot be the output of an effective FO+POLY+TC query.

Hence, as suggested by Koiran [28] and Benedikt and Libkin [5], we consider for each $\varepsilon > 0$ an ε -*volume approximation query* VOL^ε , such that for any polynomial constraint database D over \mathcal{S} , such that if $v \in \text{VOL}^\varepsilon(D)$, then

$$|v - \text{VOL}(S^D)| < \varepsilon.$$

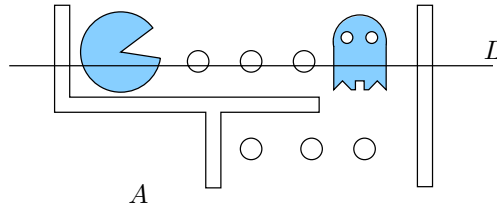


FIG. 6.7. A semialgebraic set A with $\kappa(A) = 12$.

It is known that volume approximation is not expressible in FO+POLY [5]. We show that it is expressible in FO+POLY+TC.

We will use the following result.

THEOREM 6.12 (see [28]). *Let A be a semialgebraic set in \mathbf{R}^n , and let δ -cover(A) be its box covering of size δ . Then*

$$(6.19) \quad |\text{VOL}(A) - \text{VOL}(\delta\text{-cover}(A))| < \frac{1}{\delta}(\text{diam}(A))^{n+1}\kappa(A)n,$$

where $\kappa(A)$ is the maximal number of connected components of the intersection of A with any axis-parallel line L (see Figure 6.7), and where $\text{diam}(A)$ is the diameter of A .⁵

THEOREM 6.13. *For each $\varepsilon > 0$, there exists an ε -volume approximation query in FO+POLY+TC.*

Proof. We first show that the number κ of Theorem 6.12 is expressible in FO+POLY+TC. Thereto, first we define n sets K_i which contain $(2n - 1)$ -tuples $(a_1, \dots, a_{i-1}, a_{i+1}, \dots, a_n, \vec{p})$, where $a_j \in \mathbf{R}$ for $j = 1, \dots, i - 1, i + 1, \dots, n$, and where \vec{p} is either an isolated point on the intersection of A with $\{\vec{x} \mid \bigwedge_{j \neq i} x_j = a_j\}$, or in the middle of an interval in this intersection. Using similar techniques as in section 4, we compute for each $(a_1, \dots, a_{i-1}, a_{i+1}, \dots, a_n)$ the number of points \vec{p} such that $(a_1, \dots, a_{i-1}, a_{i+1}, \dots, a_n, \vec{p}) \in K_i$. We then obtain n sets K'_i consisting of n -tuples $(a_1, \dots, a_{i-1}, a_{i+1}, \dots, a_n, N)$ with $N \in \mathbf{N}$, and we define M_i to be the maximum of all those N which are in K'_i for some $(a_1, \dots, a_{i-1}, a_{i+1}, \dots, a_n)$. Finally, $\kappa = \max\{M_1, \dots, M_n\}$.

Let $\delta = \frac{1}{\varepsilon}(\text{diam}(S^D))^n \kappa(S^D)n + 1$. By Proposition 5.14, the box covering of S^D of size δ is expressible in FO+POLY+TC. By Theorem 6.12, $\text{VOL}(\delta\text{-cover}(S^D))$ approximates the volume of S^D within an ε -error margin.

Recall that $\delta\text{-cover}(S^D)$ is represented as a $2n$ -ary relation. Each $2n$ -tuple corresponds to an n -dimensional box of size δ (see section 5.5). Let $\mathbf{nrofboxes}(y)$ be the formula

$$[\text{TC}_{\vec{b},x;\vec{b}',x'} \text{lexicographic}(\vec{b}, \vec{b}') \wedge x' = x + 1](\vec{b}_{\min}, 1, \vec{b}_{\max}, y),$$

where $\text{lexicographic}(\vec{b}, \vec{b}')$ is an FO+POLY formula expressing that \vec{b} is less than \vec{b}' with respect to the lexicographical ordering on tuples in \mathbf{R}^n , and where $\vec{b}_{\min}, \vec{b}_{\max} \in \delta\text{-cover}(S^D)$ is the minimum (respectively, maximum) n -tuple in $\delta\text{-cover}(S^D)$ with respect to the lexicographical ordering. Finally, let $N \in \mathbf{R}$ such that $\mathbf{nrofboxes}(N)$ holds. Then we define $\text{VOL}^\varepsilon(v)$ to be the FO+POLY+TC formula which expresses that $v = N\delta^n$. \square

⁵For $X \subseteq \mathbf{R}^n$ bounded, the *diameter* of X is defined as the supremum of $\{\|\vec{x} - \vec{y}\| \mid \vec{x}, \vec{y} \in X\}$.

Since the δ -approximation of A is included in the box covering $\delta\text{-cover}(A)$, a better volume approximation can be obtained by using the volume of the δ -approximation instead of the volume of $\delta\text{-cover}(A)$. By the next theorem, this also gives an FO+POLY+TC expressible ε -approximation query.

It is known that taking the volume of a semilinear set does not take us out of the semialgebraic setting and that the volume of a semilinear set can be expressed in the aggregate language FO+POLY+SUM [5].

THEOREM 6.14. *Let $\mathcal{S} = \{S\}$, with S an n -ary relation name. There exists an FO+POLY+TC formula `volume` over \mathcal{S} such that `volume`(S^D) is the volume of S^D for any linear constraint database D over \mathcal{S} .*

Proof. If $\dim(S^D) < n$, then we define `volume`(x) $\equiv x = 0$. Suppose that $\dim(S^D) = n$. Since $\text{VOL}(S^D) = \text{VOL}(\text{cl}(\text{int}(S^D)))$, we actually may assume that S^D is closed and consists entirely of n -dimensional pieces.

It is well known that S^D is a finite union of convex sets c_1, \dots, c_r of a partition of \mathbf{R}^n induced by a finite number of $(n-1)$ -dimensional hyperplanes H_1, \dots, H_s [48]. Vandeurzen, Gyssens, and Van Gucht [48] show that there exists an FO+POLY formula `hyperplanes`(v_1, \dots, v_n, d) such that `hyperplanes`(D) consists of s tuples $(\vec{v}_1, d_1), \dots, (\vec{v}_s, d_s)$ such that $H_i = \{\vec{x} \in \mathbf{R}^n \mid \vec{v}_i \vec{x} = d_i\}$. Moreover, there exists an FO+POLY formula `points` such that `points`(D) is equal to the extremal points of the convex sets c_1, \dots, c_s . Recall that the *extremal points* of a convex set are those points which cannot be written as a linear combination of two other points of the convex set [51].

We now want to retrieve the extremal points of the convex sets c_1, \dots, c_r . In order to do so, we shall first select a unique point in the interior of each convex set. With each of these points we then associate all special points which are in the corresponding convex set. These will then be the extremal points.

We thus define an FO+POLY+TC formula `unique` over \mathcal{S} such that `unique`(D) consists of points $\vec{p}_1, \dots, \vec{p}_s$ such that $\vec{p}_i \in \text{int}(c_i)$ for $i = 1, \dots, s$. The formula `unique` makes use of the following formulas over \mathcal{S} :

- A formula over \mathcal{S} which computes the barycenter of any n -dimensional simplex obtained as the convex hull of an $(n+1)$ -tuple of points in `specialpoints`(D), i.e.,

$$\text{barycenter}(\vec{x}) \equiv \exists \vec{y}_1 \cdots \exists \vec{y}_{n+1} \left(\bigwedge_{i=1}^n \text{points}(\vec{y}_i) \wedge x_i = \frac{1}{n+1} ((\vec{y}_1)_i + \cdots + (\vec{y}_{n+1})_i) \right).$$

- A formula `interiors` over \mathcal{S} which computes the interiors of the sets c_1, \dots, c_s , i.e.,

$$\text{interiors}(\vec{x}) \equiv S(\vec{x}) \wedge \neg(\exists \vec{v} \exists d (\text{hyperplanes}(\vec{v}, d) \wedge \vec{v} \cdot \vec{x} = d)).$$

- A formula over \mathcal{S} which checks whether two barycenters are in the same convex set c_i for some i , i.e.,

$$\text{samecell}(\vec{x}, \vec{y}) \equiv \text{barycenter}(\vec{x}) \wedge \text{barycenter}(\vec{y}) \wedge \forall \lambda (0 \leq \lambda \leq 1 \rightarrow \text{interiors}(\lambda \vec{x} + (1 - \lambda) \vec{y})).$$

We then define the formula $\text{unique}(\vec{x})$ as

$$\forall \vec{z} \text{samecell}(\vec{x}, \vec{z}) \rightarrow \text{lexicographic}(\vec{x}, \vec{z}),$$

where $\text{lexicographic}(\vec{x}, \vec{z})$ is an FO+POLY formula expressing that \vec{x} is less than or equal to \vec{z} with respect to the lexicographical ordering on tuples in \mathbf{R}^n .

Define the formula

$$\begin{aligned} \text{extremal}(\vec{x}, \vec{y}) \equiv & \text{points}(\vec{x}) \wedge \text{unique}(\vec{y}) \\ & \wedge \forall \lambda (0 < \lambda \leq 1) \rightarrow \text{interiors}(\lambda \vec{y} + (1 - \lambda) \vec{x}). \end{aligned}$$

We can now identify each convex set c_1, \dots, c_r , so we may focus on a single convex set. We now show that, given the extremal points of a convex set c in \mathbf{R}^n , a decomposition of c in a finite number of n -simplices can be constructed in FO+POLY. The n -simplices will be represented by $n + 1$ independent points.

We first identify the hyperplanes which have an $(n - 1)$ -dimensional intersection with the boundary of the convex set c . Let $\vec{e}_1, \dots, \vec{e}_k$ be the extremal points of c . Let onboundary be the FO+POLY formula which selects the tuples in $\text{hyperplanes}(D)$ with this property. Next, let sameface be an FO+POLY formula such that $\text{face}(\vec{e}, \vec{v}, d)$ if and only if \vec{e} is an extremal point of c , $(\vec{v}, d) \in \text{onboundary}(\vec{e}_1, \dots, \vec{e}_k)$, and $\vec{e} \in \{\vec{x} \in \mathbf{R}^n \mid \vec{v} \cdot \vec{x} = d\}$. In this way, we can group the extremal points of c such that each group corresponds to a single face of the convex cell c .

For each face of c , we now project the extremal points corresponding to this face to \mathbf{R}^{n-1} such that they are the extremal points of a convex set in \mathbf{R}^{n-1} . Thus, if $\text{face}(\vec{x}_1, \vec{v}, d, \vec{e}_1, \dots, \vec{e}_k) \wedge \dots \wedge \text{face}(\vec{x}_\ell, \vec{v}, d, \vec{e}_1, \dots, \vec{e}_k)$, then we obtain extremal points of a convex set in \mathbf{R}^{n-1} as follows: Let $i \in \{1, \dots, n\}$ be such that $\{\vec{x} \in \mathbf{R}^n \mid x_i = 0\}$ is not perpendicular to $\{\vec{x} \in \mathbf{R}^n \mid \vec{v} \cdot \vec{x} = d\}$ (this can be easily expressed in FO+POLY). Then consider the projection $\pi_i : \mathbf{R}^n \rightarrow \mathbf{R}^{n-1}$ defined as $\pi_i(x_1, \dots, x_n) \mapsto (x_1, \dots, x_{i-1}, x_{i+1}, \dots, x_n)$ and apply this map on $\vec{x}_1, \dots, \vec{x}_\ell$.

Algorithm TRIANGULATE-IN- n -DIMENSIONS

Input: The extremal points $\vec{e}_1, \dots, \vec{e}_k$ of a convex set c in \mathbf{R}^n .
Output: A finite number of n -simplices forming a decomposition of c .
Method:

1. Compute the pairs $(\vec{v}, d) \in \text{onboundary}(\vec{e}_1, \dots, \vec{e}_k)$.
2. For each $(\vec{v}, d) \in \text{onboundary}(\vec{e}_1, \dots, \vec{e}_k)$ do the following:
 - (a) Compute $\text{face}(\vec{x}, \vec{v}, d, \vec{e}_1, \dots, \vec{e}_k)$.
 - (b) Find an i as described above and call TRIANGULATE-IN- $(n - 1)$ -DIMENSIONS($\pi_i(\text{face}(\vec{v}, d, \vec{e}_1, \dots, \vec{e}_k))$).
3. Select a point \vec{p}_{n+1} in the interior of c .
4. Output the $(n + 1)$ -tuples $(\vec{p}_1, \dots, \vec{p}_n, \vec{p}_{n+1})$, where $(\vec{p}_1, \dots, \vec{p}_n)$ is an n -tuple in the result of the calls of TRIANGULATE-IN- $(n - 1)$ -DIMENSIONS in step 2(b).

We now define the FO+POLY formula simplexdecomp over \mathcal{S} such that $\text{simplexdecomp}(D)$ is a decomposition into n -simplices of S^D for any polynomial constraint database D over $\{S\}$. Let triang be a formula which expresses the algorithm TRIANGULATE-IN- n -DIMENSIONS. Then

$$\begin{aligned} \text{simplexdecomp}(\vec{x}_1, \dots, \vec{x}_{n+1}) \equiv & \exists \vec{y} (\text{unique}(\vec{y}) \\ & \wedge \text{triang}(\text{extremal})(\vec{x}_1, \dots, \vec{x}_{n+1}, \vec{y})). \end{aligned}$$

Let $(\vec{p}_1, \dots, \vec{p}_{n+1})$ be an n -simplex points. Let $\vec{r}_i = \vec{p}_i - \vec{p}_1$ for $i = 2, \dots, n + 1$, and let G be the $n \times n$ matrix whose rows contain the coordinates of the vectors \vec{r}_j for $1 \leq j \leq n$. Then by the Gram determinant formula [37], the volume of $(\vec{p}_1, \dots, \vec{p}_{n+1})$ is equal to

$$\frac{|\det(GG^t)|^{\frac{1}{2}}}{n!},$$

where G^t is the transpose of G . Hence, the volumes of the simplices are expressible by an FO+POLY formula, which we will denote by `volsimplex`.

Finally, define

$$\begin{aligned} \Psi(y) \equiv & [\text{TC}_{x,s;x',s'} s = \exists \vec{p}_1, \dots, \exists \vec{p}_{n+1}, \exists \vec{q}_1, \dots, \exists \vec{q}_{n+1} \\ & \text{volsimplex}(\vec{p}_1, \dots, \vec{p}_{n+1}) \wedge s' = \text{volsimplex}(\vec{q}_1, \dots, \vec{q}_{n+1}) \\ & \wedge \text{successor}(\vec{q}_1, \dots, \vec{q}_{n+1}, \vec{p}_1, \dots, \vec{p}_{n+1}) \\ & \wedge \text{simplexdecomp}(\vec{p}_1, \dots, \vec{p}_{n+1}) \wedge \text{simplexdecomp}(\vec{q}_1, \dots, \vec{q}_{n+1}) \\ & \wedge x' = x + s](0, v_1, y, v_\ell), \end{aligned}$$

where `successor` is a successor relation defined on the n -simplices in the decomposition into simplices `simplexdecomp`(D), and where v_1 and v_ℓ are, respectively, the volume of the first and last simplex according to this successor relation. The total volume of S^D is then given by

$$\text{volume}(v) \equiv \exists y \Psi(y) \wedge v = y + v_\ell,$$

with v_ℓ as above. □

Acknowledgments. We would like to thank the referees for their comments which contributed significantly towards the readability of the paper.

REFERENCES

- [1] D. ABEL AND B. C. OOI, EDs., *Advances in Spatial Databases—3rd Symposium (SSD'93)*, Lecture Notes in Comput. Sci. 692, Springer-Verlag, Berlin, New York, 1993.
- [2] S. BASU, R. POLLACK, AND M.-F. ROY, *On the combinatorial and algebraic complexity of quantifier elimination*, J. ACM, 43 (1996), pp. 1002–1046.
- [3] S. BASU, R. POLLACK, AND M.-F. ROY, *Algorithms in Real Algebraic Geometry*, Algorithms Comput. Math. 10, Springer-Verlag, Berlin, New York, 2003.
- [4] M. BENEDIKT AND L. LIBKIN, *Safe constraint queries*, SIAM J. Comput., 29 (2000), pp. 1652–1682.
- [5] M. BENEDIKT AND L. LIBKIN, *Aggregate operators in constraint query languages*, J. Comput. System Sci., 64 (2002), pp. 626–654.
- [6] J. BOCHNAK, M. COSTE, AND M.-F. ROY, *Real Algebraic Geometry*, *Ergeb. Math. Grenzgeb.* (3) 36, Springer-Verlag, Berlin, 1998.
- [7] H. BRAKHAGE, *Topologische Eigenschaften algebraischer Gebilde über einem beliebigen reell-abgeschlossenen Konstantenkörper*, Dissertation, Universität Heidelberg, Heidelberg, Germany, 1954.
- [8] A. P. BUCHMANN, O. GÜNTHER, T. R. SMITH, AND Y.-F. WANG, EDs., *Design and Implementation of Large Spatial Databases—First Symposium (SSD'89)*, Lecture Notes in Comput. Sci. 409, Springer-Verlag, Berlin, New York, 1989.
- [9] M. DE BERG, M. VAN KREVELD, M. OVERMARS, AND O. SCHWARKOPF, *Computational Geometry*, Springer-Verlag, Berlin, New York, 1997.
- [10] F. DUMORTIER, M. GYSSENS, L. VANDEURZEN, AND D. VAN GUCHT, *On the decidability of semilinearity for semialgebraic sets and its implications for spatial databases*, J. Comput. System Sci., 58 (1999), pp. 535–571.

- [11] H. D. EBBINGHAUS AND J. FLUM, *Finite Model Theory*, Springer-Verlag, Berlin, 1995.
- [12] M. J. EGENHOFER AND J. R. HERRING, EDs., *Advances in Spatial Databases—4th Symposium (SSD'95)*, Lecture Notes in Comput. Sci. 951, Springer-Verlag, Berlin, New York, 1995.
- [13] F. GEERTS, *Linear approximation of semi-algebraic spatial databases using transitive closure logic, in arbitrary dimension*, in Proceedings of the 8th International Workshop on Databases and Programming Languages, Lecture Notes in Comput. Sci., Springer-Verlag, 2002, pp.
- [14] F. GEERTS, *Expressing the box cone radius in the relational calculus with real polynomial constraints*, Discrete Comput. Geom., 30 (2003), pp. 607–622.
- [15] F. GEERTS AND B. KUIJPERS, *Expressing topological connectivity of spatial databases*, in Research Issues in Structured and Semistructured Database Programming, Proceedings of the 7th International Workshop on Database Programming Languages, Lecture Notes in Comput. Sci. 1949, Springer-Verlag, Berlin, New York, 1999, pp. 224–238.
- [16] F. GEERTS AND B. KUIJPERS, *Linear approximation of planar spatial databases using transitive-closure logic*, in Proceedings of the 19th ACM Symposium on Principles of Database Systems, ACM, New York, 2000, pp. 126–135.
- [17] F. GEERTS AND B. KUIJPERS, *On the decidability of termination of query evaluation in transitive-closure logics for polynomial constraint databases*, Theoret. Comput. Sci., 336 (2005), pp. 125–151.
- [18] S. GRUMBACH AND G. KUPER, *Tractable recursion over geometric data*, in Proceedings of the 3rd Conference on Principles and Practice of Constraint Programming, G. Smolka, ed., Lecture Notes in Comput. Sci. 1330, Springer-Verlag, Berlin, New York, 1997, pp. 450–462.
- [19] S. GRUMBACH, P. RIGAUX, M. SCHOLL, AND L. SEGOUFIN, *DEDALE, a spatial constraint database*, in Proceedings of Database Programming Languages (DBPL '97), S. Cluet and R. Hull, eds., Lecture Notes in Comput. Sci. 1369, Springer-Verlag, Berlin, New York, 1998, pp. 38–59.
- [20] S. GRUMBACH, P. RIGAUX, AND L. SEGOUFIN, *The DEDALE system for complex spatial queries*, in Proceedings of the ACM International Conference on Management of Data (SIGMOD '98), L. M. Haas and A. Tiwary, eds., ACM, New York, 1998, pp. 213–224.
- [21] S. GRUMBACH AND J. SU, *Towards practical constraint databases*, in Proceedings of the 15th ACM Symposium on Principles of Database Systems (PODS '96), ACM, New York, 1996, pp. 28–39.
- [22] S. GRUMBACH AND J. SU, *Queries with arithmetical constraints*, Theoret. Comput. Sci., 173 (1997), pp. 151–181.
- [23] V. GUILLEMIN AND A. POLLACK, *Differential Topology*, Prentice-Hall, Englewood Cliffs, NJ, 1974.
- [24] O. GUNTHER AND H.-J. SCHEK, EDs., *Advances in Spatial Databases—2nd Symposium (SSD'91)*, Lecture Notes in Comput. Sci. 525, Springer-Verlag, Berlin, New York, 1991.
- [25] R. H. GÜTING, ED., *Advances in Spatial Databases—6th Symposium (SSD'99)*, Lecture Notes in Comput. Sci. 1651, Springer-Verlag, Berlin, New York, 1999.
- [26] R. HARDT, *Triangulation of subanalytic sets and proper light subanalytic maps*, Invent. Math., 38 (1976/77), pp. 207–217.
- [27] P. C. KANELLAKIS, G. M. KUPER, AND P. Z. REVESZ, *Constraint query languages*, J. Comput. System Sci., 51 (1995), pp. 26–52.
- [28] P. KOIRAN, *Approximating the volume of definable sets*, in Proceedings of the 36th IEEE Symposium on Foundations of Computer Science (FOCS), IEEE, Los Alamitos, CA, 1995, pp. 134–141.
- [29] S. KREUTZER, *Operational semantics for fixed-point logics on constraint databases*, in Proceedings of the 8th International Conference on Logic for Programming, Artificial Intelligence, and Reasoning, R. Nieuwenhuis and A. Voronkov, eds., Lecture Notes in Comput. Sci. 2250, Springer, Berlin, 2002, pp. 470–484.
- [30] S. KREUTZER, *Query languages for constraint databases: First-order logic, fixed-points, and convex hulls*, in Proceedings of the 9th International Conference on Database Theory, J. Van den Bussche and V. Vianu, eds., Lecture Notes in Comput. Sci. 1973, Springer-Verlag, Berlin, New York, 2001, pp. 248–262.
- [31] B. KUIJPERS, J. PAREDAENS, M. SMITS, AND J. VAN DEN BUSSCHE, *Termination properties of spatial datalog*, in Logic in Databases (LID '96), D. Pedreschi and C. Zaniolo, eds., Lecture Notes in Comput. Sci. 1154, Springer-Verlag, Berlin, New York, 1996, pp. 101–116.
- [32] B. KUIJPERS, J. PAREDAENS, AND J. VAN DEN BUSSCHE, *Topological elementary equivalence of closed semi-algebraic sets in the real plane*, J. Symbolic Logic, 65 (2000), pp. 1530–1555.

- [33] B. KUIJPERS AND M. SMITS, *On expressing topological connectivity in spatial datalog*, in Proceedings of the 2nd Workshop on Constraint Databases and Applications, V. Gaede, A. Brodsky, O. Gunter, D. Srivastava, V. Vianu, and M. Wallace, eds., Lecture Notes in Comput. Sci. 1191, Springer-Verlag, Berlin, New York, 1997, pp. 116–133.
- [34] G. M. KUPER, J. PAREDAENS, AND L. LIBKIN, EDS., *Constraint Databases*, Springer-Verlag, Berlin, New York, 2000.
- [35] J. M. LEE, *Introduction to Topological Manifolds*, Graduate Texts in Math. 202, Springer-Verlag, New York, 2000.
- [36] E. E. MOISE, *Geometrical Topology in Dimensions 2 and 3*, Springer-Verlag, Berlin, New York, 1977.
- [37] J. O’ROURKE, *Computational Geometry in C*, Cambridge University Press, Cambridge, UK, 1998.
- [38] F. P. PREPARATA AND M. I. SHAMOS, *Computational Geometry—An Introduction*, Springer-Verlag, Berlin, New York, 1985.
- [39] E. RANNOU, *The complexity of stratification computation*, Discrete Comput. Geom., 19 (1998), pp. 47–79.
- [40] E. RANNOU, *Personal communication*, 2000.
- [41] J. RENEGAR, *On the computational complexity and geometry of the first-order theory of the reals*, J. Symbolic Comput., 13 (1992), pp. 255–352.
- [42] M. SCHOLL AND A. VOISARD, EDS., *Advances in Spatial Databases—5th Symposium (SSD’97)*, Lecture Notes in Comput. Sci. 1262, Springer-Verlag, Berlin, New York, 1997.
- [43] A. TARSKI, *A Decision Method for Elementary Algebra and Geometry*, 2nd ed., University of California Press, Berkeley and Los Angeles, 1951.
- [44] L. VAN DEN DRIES, *Tame Topology and O-minimal Structures*, Cambridge University Press, Cambridge, UK, 1998.
- [45] L. VAN DEN DRIES AND C. MILLER, *Geometric categories and O-minimal structures*, Duke Math. J., 82 (1996), pp. 497–540.
- [46] L. VANDEURZEN, *Logic-Based Query Languages for the Linear Constraint Database Model*, Ph.D. thesis, Limburgs Universitair Centrum (LUC), Diepenbeek, Belgium, 1999.
- [47] L. VANDEURZEN, M. GYSENS, AND D. VAN GUCHT, *On the desirability and limitations of linear spatial query languages*, in Advances in Spatial Databases—4th Symposium (SSD ’95), M. J. Egenhofer and J. R. Herring, eds., Lecture Notes in Comput. Sci. 951, Springer-Verlag, Berlin, New York, 1995, pp. 14–28.
- [48] L. VANDEURZEN, M. GYSENS, AND D. VAN GUCHT, *An expressive language for linear spatial database queries*, in Proceedings of the 17th ACM Symposium on Principles of Database Systems (PODS ’98), ACM, New York, 1998, pp. 109–118.
- [49] A. J. WILKIE, *On defining C^∞* , J. Symbolic Logic, 59 (1994), p. 344.
- [50] A. J. WILKIE, *A theorem of the complement and some new o-minimal structures*, Selecta Math. (N. S.), 5 (1999), pp. 397–421.
- [51] G. M. ZIEGLER, *Lectures on Polytopes*, Graduate Texts in Math. 152, Springer-Verlag, Berlin, New York, 1998.

PARTIAL MATCH QUERIES IN RANDOM k -d TREES*

HUA-HUAI CHERN[†] AND HSIEN-KUEI HWANG[‡]

Abstract. We solve the open problem of characterizing the leading constant in the asymptotic approximation to the expected cost used for random partial match queries in random k -d trees. Our approach is new and of some generality; in particular, it is applicable to many problems involving differential equations (or difference equations) with polynomial coefficients.

Key words. k -d trees, partial match queries, differential equations, average-case analysis of algorithms, method of linear operators, asymptotic analysis

AMS subject classifications. 68W40, 68P05, 68P10, 68U05

DOI. 10.1137/S0097539703437491

1. Introduction. Multidimensional binary search trees (abbreviated as k -d trees or simply kd trees, k being the dimensionality) were first proposed by Bentley [2] and represent extremely useful data structures for problems in diverse fields, especially those having to do with range queries; nearest neighbor search; or partial, exact, or approximate match queries. For example, they are useful in statistical learning, databases, data-mining, computer graphics, robotics, medical imaging, neural networks, multimedia, statistical computing, computer-aided design, astronomy, pattern recognition, geographic information systems, music information retrieval, computational biology, etc. For more information, see Bentley [3], Bentley and Friedman [4], Bertino et al. [5], Friedman, Bentley, and Finkel [19], Gray and Moore [21], Grother, Candela, and Blue [22], Nichol et al. [28], Omohundro [29], Preparata and Shamos [31], Reiss, Aucouturier, and Sandler [32], Samet [33, 34], and Schwarzer and Lotan [35]. According to the statistics collected in the Stony Brook Algorithm Repository, k -d trees were among the most popular algorithmic problems; see Skiena’s account in [37].

Despite the usefulness and diversity of k -d trees, their precise probabilistic analysis appears only sporadically in the literature (although many properties can be derived from those for random binary search trees); see [6, 11, 12, 13, 19, 20, 25, 26, 27]. We are concerned in this paper with the expected cost used by random partial match queries in random k -d trees. The growth order of this cost has been known since Flajolet and Puech’s work [17], but the characterization of the leading constant remains a very challenging problem. We propose a new approach to filling this gap.

The prototypes of k -d trees are binary search trees when there is a total ordering for the input keys or when $k = 1$. A binary search tree \mathcal{B} is a binary tree constructed from a given sequence of keys as follows. If the tree size is $n = 0$, then \mathcal{B} is empty. If $n \geq 1$, then the first key is placed at the root. The remaining keys are compared successively to the root key, which may be called a “discriminator,” and are directed to the left (or right) branch if they are smaller (or larger); keys directed to the same

*Received by the editors November 10, 2003; accepted for publication (in revised form) December 8, 2005; published electronically April 7, 2006.

<http://www.siam.org/journals/sicomp/35-6/43749.html>

[†]Department of Computer Science, National Taiwan Ocean University, 2 Pei-Ning Road, Keelung 20224, Taiwan (felix@cs.ntou.edu.tw).

[‡]Institute of Statistical Science, Academia Sinica, Taipei 115, Taiwan (hkhwang@stat.sinica.edu.tw).

branch are constructed recursively as a binary search tree. By construction, a query operation such as “ $x \in \mathcal{B}$?” can be easily carried out in binary search trees; hence the name “search tree.”

When given k -dimensional points or keys, $k \geq 2$, the simple idea of k -d trees is to use each coordinate cyclically, say, in the order from the first coordinate to the last, as the “discriminator” to direct points falling in the same subtree as in binary search trees (but using the $(\ell \bmod k) + 1$ st coordinate for a node at a distance of ℓ from the root); see Figure 1.1 for a plot of a simple 2-d tree.

In addition to exact match searches, as in binary search trees, k -d trees can also be used for *partial match queries* when some coordinates are either unspecified, don’t-cares, or wild cards. Thus the range search is continued in both subtrees when the unspecified coordinate is used as a discriminator in the k -d tree and in either the left or the right subtree (but not both) otherwise; see Bentley [2] for details. Two simple instances of a range search in 2-d trees are shown in Figure 1.2.

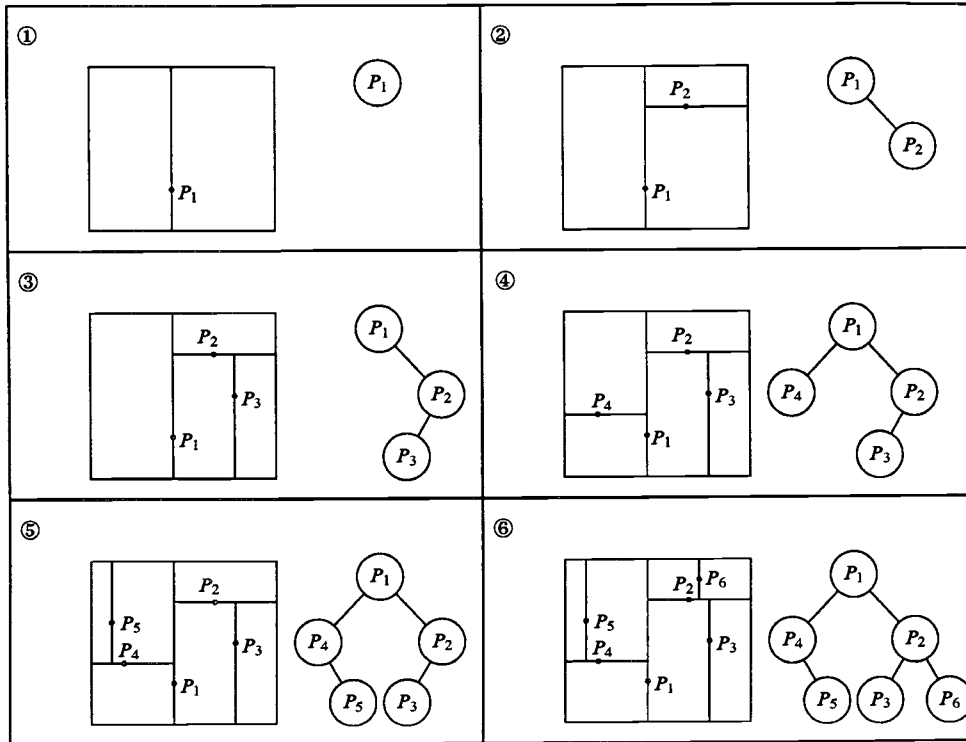


FIG. 1.1. The stepwise constructions of a 2-d tree of six points.

We consider the following probabilistic model for our average-case analysis. A sequence of n independently and identically distributed random points in $[0, 1]^k$ is given, where each coordinate is uniformly distributed over the unit interval and is independent of other coordinates. Then we construct the k -d tree \mathcal{T} , called a *random k -d tree*. For partial match queries, we then consider a query of the form $\mathcal{Y} = (Y_1, \dots, Y_k)$, where Y_j is either the unit interval (meaning don’t-care) or a random variable uniformly distributed over the unit interval ($Y_j = \text{Uniform}(0, 1)$), the number of specified coordinates s satisfying $0 \leq s \leq k$. We then perform the range search of

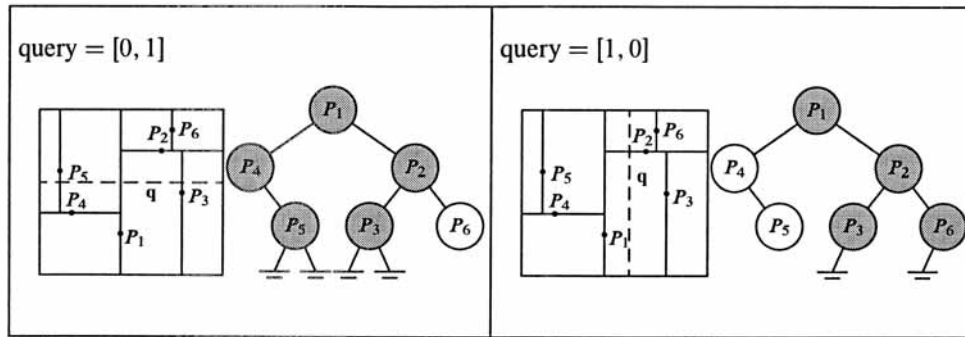


FIG. 1.2. Two instances of partial match queries (as marked by the dashed line) in a 2-d tree: visited nodes are the filled-in grey circles. Here the query pattern $[0, 1]$ means that the first coordinate is unspecified and the second is specified; the meaning of $[1, 0]$ is similar.

the query \mathcal{Y} in the tree \mathcal{T} . The number of nodes visited is a random variable, say, X_n . The main quantity of interest in this paper is the expected value $Q_n := \mathbb{E}(X_n)$, or simply the *expected cost of a random partial match query in random k -d trees of n nodes*.

While the uniform model we are considering may seem too idealized, it is simple yet mathematically tractable; also the asymptotics obtained under such a model usually subsists under more general ones; see [17] for more discussions.

For convenience, throughout this paper we write the query pattern as $\mathbf{q} := [q_1, \dots, q_k]$ since Q_n depends only on \mathbf{q} , where $q_j \in \{0, 1\}$ and $0 \leq q_1 + \dots + q_k \leq k$. Here $q_j = 0$ means that the j th coordinate is unspecified, and $q_j = 1$ means that it is specified. Flajolet and Puech [17] showed that

$$(1.1) \quad Q_n \sim Cn^{\alpha-1} \quad (1 \leq s < k),$$

where C is a constant and $\alpha > 1$ solves the equation

$$(\alpha + q_1) \cdots (\alpha + q_k) = 2^k,$$

or, equivalently, the equation $\alpha^{k-s}(\alpha + 1)^s = 2^k$, where $s = q_1 + \dots + q_k$ denotes the number of specified coordinates. Their result corrected the original claim of Bentley [2, p. 513] that $Q_n = O(n^{1-s/k})$ since α can be written as $\alpha = 1 - s/k + \varepsilon(s/k)$, where $\varepsilon(u) > 0$ for $0 < u < 1$; see [17] and [6].

The approach of Flajolet and Puech [17] is based on a linear differential system and starts from the generating function $y_1(z) := \sum_{n \geq 1} (n + 1)Q_n z^n$. Then

$$\frac{d}{dz} \mathbf{y}(z) = \Omega(z) \mathbf{y}(z) + \mathbf{b}(z),$$

where $\mathbf{y}(z) = (y_1(z), \dots, y_{2k-s}(z))^T$, $\mathbf{b}(z) = (b_1(z), \dots, b_{2k-s}(z))^T$, and $\Omega(z)$ is a $(2k - s) \times (2k - s)$ matrix. For example, when $\mathbf{q} = [0, 1, 1]$,

$$\frac{d}{dz} \begin{pmatrix} y_1(z) \\ y_2(z) \\ y_3(z) \\ y_4(z) \end{pmatrix} = \begin{pmatrix} \frac{1}{z(1-z)} & \frac{2}{1-z} & 0 & -\frac{1}{z^2(1-z)} \\ 0 & 0 & \frac{2}{1-z} & 0 \\ \frac{2}{1-z} & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{pmatrix} \begin{pmatrix} y_1(z) \\ y_2(z) \\ y_3(z) \\ y_4(z) \end{pmatrix} + \begin{pmatrix} \frac{1}{(1-z)^3} \\ \frac{2}{(1-z)^3} \\ \frac{2}{(1-z)^3} \\ 0 \end{pmatrix}.$$

From this system, they proved that the generating function $Q(z) := \sum_{n \geq 1} Q_n z^n$ satisfies

$$Q(z) \sim K(1 - z)^{-\alpha}$$

for some constant K and for z in some region in the z -plane. Then (1.1) follows from singularity analysis (see [16]). But as is often the case for linear systems, the constant K or C in (1.1) does not seem to have simple computable forms by this approach.

Chanzy, Devroye, and Zamora-Cura [6] proved (1.1) by a more combinatorial and probabilistic approach, but the calculation of C still remains open.

The recurrence for Q_n we are solving is of the form ($j_0 := n$)

$$(1.2) \quad \begin{aligned} Q_n = 1 + & \sum_{1 \leq i < k} 2^i \sum_{1 \leq j_i < \dots < j_1 < n} \prod_{1 \leq r \leq i} \frac{1 + q_r j_r}{j_{r-1}(1 + q_r j_{r-1})} \\ & + 2^k \sum_{1 \leq j_k < \dots < j_1 < n} Q_{j_k} \prod_{1 \leq r \leq k} \frac{1 + q_r j_r}{j_{r-1}(1 + q_r j_{r-1})} \end{aligned}$$

for $n \geq 1$, with $Q_0 := 0$. In particular (writing $Q_n = Q_n[q_1, \dots, q_k]$)

$$\begin{aligned} Q_n[0, 1] &= 3 - \frac{2}{n} + \frac{4}{n^2} \sum_{1 \leq j < n} (n - 1 - j) Q_j[0, 1], \\ Q_n[0, 0, 1] &= 7 - \frac{6}{n} - \frac{4}{n} H_{n-1} \\ &\quad + \frac{8}{n} \sum_{1 \leq j < n} \left(H_{n-1} - H_j - (j + 1) \left(H_{n-1}^{(2)} - H_j^{(2)} \right) \right) Q_j[0, 0, 1] \end{aligned}$$

for $n \geq 1$ with $Q_0 := 0$. Here $H_n^{(i)} := \sum_{1 \leq j \leq n} j^{-i}$ and $H_n = H_n^{(1)}$.

While the problem we are studying in this paper is similar to partial match queries in random quadtrees (see [14, 9]), the nature of the associated analytic problems is very different (although both problems can be written in terms of linear systems). Intuitively, random quadtrees seem to have more independence in subtrees, while the independence of k -d trees is somehow “bound” by the cyclic construction. For partial match queries in quadtrees, the dominant asymptotic approximation to the expected cost depends only on the number (but not the pattern) of specified coordinates; in contrast, the leading constant in the asymptotic approximation to the expected cost for k -d trees depends strongly on the query pattern, making the analytic problem harder.

We propose in the next section a more straightforward approach, which is based on translating the system of recurrences satisfied by Q_n into a *scalar differential equation with polynomial coefficients* for the generating function of Q_n . Some explicitly solvable cases of the differential equation are discussed in section 3. We then solve the general differential equation by using Mellin transforms (see Flajolet, Gourdon, and Dumas [15]) in section 4. This Mellin approach leads, by suitably changing the initial conditions, to a series form for K that is absolutely convergent. But the justification of the application of the Mellin inversion integrals is more delicate and requires stronger analytic estimates. We thus develop in section 5 a different approach based on extending our asymptotic theory for Cauchy–Euler differential equations (see [10]). Such an approach is very general and is mostly algebraic and elementary in nature, requiring little knowledge in differential equations as in [10]. It is also applicable to

other situations, where one has linear differential equations with polynomial coefficients. We briefly indicate the application to k -d- t trees in section 6, which are locally balanced versions of k -d trees; see [11]. The development of a general theory with many applications will be given elsewhere; see [7] for further exploration of the same approach to quadtrees.

Notation. Throughout this paper, $[q_1, \dots, q_k] \in \{0, 1\}^k$, $k \geq 2$, always denotes the query pattern with $s := q_1 + \dots + q_k$, where 0 means unspecified and 1 specified. We also introduce the symbol

$$\rho_{n,j}(q) := \frac{1 + qj}{1 + qn} \quad (q \in \{0, 1\}).$$

Finally, $\alpha > 1$ denotes the zero of the equation (except in section 6 on k -d- t trees)

$$(x + q_1) \cdots (x + q_k) - 2^k = x^{k-s}(x + 1)^s - 2^k = 0.$$

2. From recurrences to differential equations. In this section, we first show that the expected cost $Q_n = Q_n[q_1, \dots, q_k]$ satisfies a system of recurrences (of quick-sort type) and then derive a “normal form” for the differential equation satisfied by the generating function $Q(z) = Q[q_1, \dots, q_k](z) := \sum_{n \geq 1} Q_n z^n$.

Basic recurrences. By definition, $Q_0 = 0$ for all query patterns.

LEMMA 2.1. *The expected cost Q_n satisfies the recurrence*

$$(2.1) \quad Q_n[q_1, \dots, q_k] = 1 + \frac{2}{n} \sum_{1 \leq j < n} \rho_{n,j}(q_1) Q_j[q_2, \dots, q_k, q_1] \quad (n \geq 1),$$

with $Q_0 = 0$.

Proof. When the first coordinate is unspecified, the search is conducted in both subtrees. The probability that the left subtree is of size j is $1/n$ for $0 \leq j < n$. By cyclic construction of k -d trees, we then have

$$Q_n[q_1, \dots, q_k] = 1 + \frac{1}{n} \sum_{0 \leq j < n} (Q_j[q_2, \dots, q_k, q_1] + Q_{n-1-j}[q_2, \dots, q_k, q_1]);$$

thus (2.1) follows with $q_1 = 0$. On the other hand, if $q_1 = 1$, then the probability of going to the left subtree of the root (when it has j nodes) is $(j + 1)/(n + 1)$ since with probability 1 the query results in an unsuccessful search. Thus

$$Q_n[q_1, \dots, q_k] = 1 + \frac{1}{n} \sum_{0 \leq j < n} \left(\frac{j + 1}{n + 1} Q_j[q_2, \dots, q_k, q_1] + \frac{n - j}{n + 1} Q_{n-1-j}[q_2, \dots, q_k, q_1] \right).$$

This proves (2.1). \square

Write $Q_n^{[j]} := Q_n[q_j, \dots, q_k, q_1, \dots, q_{j-1}]$, $1 \leq j \leq k$.

COROLLARY 2.2. *The $Q_n^{[j]}$'s satisfy the system of recurrences*

$$(2.2) \quad \begin{cases} Q_n^{[1]} = 1 + \frac{2}{n} \sum_{1 \leq j < n} \rho_{n,j}(q_1) Q_j^{[2]}, \\ \vdots \\ Q_n^{[k]} = 1 + \frac{2}{n} \sum_{1 \leq j < n} \rho_{n,j}(q_k) Q_j^{[1]}, \end{cases} \quad (n \geq 1).$$

Proof. The proof follows from Lemma 2.1 and the cyclic relation of k -d trees. \square

The recurrence (1.2) then follows from (2.2).

COROLLARY 2.3. *If we assume that $Q_n^{[1]} \sim Cn^{\beta-1}$ for $\beta > 1$, then*

$$(2.3) \quad Q_n^{[j]} \sim \frac{2^{k-j+1}C}{(\beta + q_j) \cdots (\beta + q_k)} n^{\beta-1} \quad (j = 1, \dots, k),$$

and β solves the equation $(\beta + q_1) \cdots (\beta + q_k) = 2^k$.

Proof. From (2.2), we have

$$Q_n^{[k]} \sim \frac{2C}{n^{1+q_k}} \sum_{1 \leq j < n} j^{\beta-1+q_k} \sim \frac{2C}{\beta + q_k} n^{\beta-1}.$$

Then the asymptotics of $Q_n^{[k-j]}$ can be obtained one after another. On the other hand, since

$$Q_n^{[1]} \sim \frac{2^k C}{(\beta + q_1) \cdots (\beta + q_k)} n^{\beta-1} \sim Cn^{\beta-1},$$

β has to satisfy the equation $(\beta + q_1) \cdots (\beta + q_k) = 2^k$. Thus $\beta = \alpha$. \square

The corollary says that once we obtain the asymptotics of one of the $Q_n^{[j]}$'s, those of the others are all known. One can show that the number of distinct query patterns needed to be computed in order to derive Q_n for all 2^k possible queries is equal to (essentially the number of different types of necklaces with two distinct colors)

$$(2.4) \quad N_k := k^{-1} \sum_{d|k} \phi(d) 2^{k/d},$$

where $\phi(d)$ denotes Euler's totient function; see Flajolet and Sedgewick [18, p. 18]. For large k , $N_k \sim 2^k/k$.

We can rewrite and simplify (2.2) as follows.

COROLLARY 2.4. *For $n \geq 1$,*

$$(2.5) \quad \begin{pmatrix} Q_n^{[1]} \\ \vdots \\ Q_n^{[k]} \end{pmatrix} = \mathbf{M} \begin{pmatrix} Q_{n-1}^{[1]} \\ \vdots \\ Q_{n-1}^{[k]} \end{pmatrix} + \begin{pmatrix} \frac{1+q_1}{n+q_1} \\ \vdots \\ \frac{1+q_k}{n+q_k} \end{pmatrix},$$

where $\mathbf{M} = (a_{i,j})_{k \times k}$ with $a_{i,i} = (n-1)/(n+q_i)$, $a_{i,(i+1) \bmod k} = 2/(n+q_i)$, and all other entries are zeros.

Proof. Taking the difference $n(n+q_j)Q_n^{[j]} - (n-1)(n-1-q_j)Q_{n-1}^{[j]}$ gives

$$(n+q_j)Q_n^{[j]} = (n-1)Q_{n-1}^{[j]} + 2Q_{n-1}^{[j+1]} + 1 + q_j \quad (1 \leq j \leq k),$$

where $Q_n^{[k+1]} = Q_n^{[1]}$. \square

The nonhomogeneous matrix recurrence (2.5) can be converted into a homogeneous one by considering

$$\bar{Q}_n^{[j]} := Q_n^{[j]} + \frac{1 + \sum_{1 \leq \ell < k} (2 - q_1) \cdots (2 - q_\ell)}{2^s - 1};$$

then

$$\begin{pmatrix} \bar{Q}_n^{[1]} \\ \vdots \\ \bar{Q}_n^{[k]} \end{pmatrix} = \mathbf{M} \begin{pmatrix} \bar{Q}_{n-1}^{[1]} \\ \vdots \\ \bar{Q}_{n-1}^{[k]} \end{pmatrix},$$

with obvious initial conditions.

Integral equations. Lemma 2.1 is next translated into an integral equation for the generating function $Q(z)$. For convenience, we introduce two integral operators,

$$\begin{cases} I_0[\varphi](z) := \int_0^z \frac{\varphi(t)}{1-t} dt, \\ I_1[\varphi](z) := \frac{1-z}{z} \int_0^z \frac{t\varphi(t)}{(1-t)^2} dt. \end{cases}$$

Let $f_j(z) = Q[q_j, \dots, q_k, q_1, \dots, q_{j-1}](z)$, $1 \leq j \leq k$.

LEMMA 2.5. *The generating function f_1 satisfies the integral equation*

$$(2.6) \quad f_1(z) = \frac{z}{1-z} + 2I_{q_1}[f_2](z).$$

Proof. The proof for (2.6) with $q_1 = 0$ is straightforward. When $q_1 = 1$, (2.6) follows from

$$\begin{aligned} \sum_{n \geq 1} \frac{z^n}{n(n+1)} \sum_{0 \leq j < n} (j+1)Q_j &= \sum_{n \geq 1} \left(\frac{z^n}{n} - \frac{z^n}{n+1} \right) \sum_{0 \leq j < n} (j+1)Q_j \\ &= \int_0^z \frac{1}{1-t} (tQ(t))' dt - \frac{1}{z} \int_0^z \frac{t}{1-t} (tQ(t))' dt. \quad \square \end{aligned}$$

The form given in (2.6) for $q_1 = 1$ is the main diverging point on which our approach differs from that used in [17]. Roughly, we keep the inverse operator of I_1 to be of first order, so that the associated linear system is of degree k , instead of $2k - s$ as used in [17]; cf. also (2.5).

COROLLARY 2.6. *We have*

$$\begin{cases} f_1(z) = 2I_{q_1}[f_2](z) + \frac{z}{1-z}, \\ \vdots \\ f_k(z) = 2I_{q_k}[f_1](z) + \frac{z}{1-z}. \end{cases}$$

By composing the above integral equations, we obtain a single integral equation for $f_1(z) = Q(z)$.

COROLLARY 2.7. *The generating function of Q_n satisfies*

$$(2.7) \quad Q(z) - 2^k(I_{q_1} \circ \dots \circ I_{q_k})[Q](z) = G_0(z),$$

where $(I_{q_1} \circ \dots \circ I_{q_j})[f](z) := I_{q_1}[\dots[I_{q_j}[f]]\dots](z)$, $1 \leq j \leq k$, and

$$G_0(z) = \frac{z}{1-z} + \sum_{1 \leq j < k} 2^j(I_{q_1} \circ \dots \circ I_{q_j}) \left[\frac{z}{1-z} \right] (z).$$

Note that $Q_n = [z^n]G_0(z)$ for $1 \leq n < k$, where $[z^n]\varphi(z)$ denotes the coefficient of z^n in the Taylor expansion of φ . On the other hand, by (1.2),

$$Q_n = 1 + \sum_{1 \leq i \leq n} 2^i \sum_{1 \leq j_i < \dots < j_1 < n} \prod_{1 \leq r \leq i} \frac{1 + q_r j_r}{j_{r-1}(1 + q_r j_{r-1})} \quad (1 \leq n < k).$$

Differential equations. Define the differential operator $\vartheta := (1 - z)(d/dz)$. Then the inverse operator of I_q is

$$I_q^{-1} = \vartheta + \frac{q}{z} \quad (q \in \{0, 1\}),$$

and the integral equation (2.7) can be converted into a scalar differential equation

$$(2.8) \quad \left(\vartheta + \frac{q_k}{z}\right) \dots \left(\vartheta + \frac{q_1}{z}\right) Q(z) - 2^k Q(z) = \sum_{1 \leq j \leq k} 2^{j-1} \left(\vartheta + \frac{q_k}{z}\right) \dots \left(\vartheta + \frac{q_j}{z}\right) \frac{z}{1-z}.$$

If we multiply both sides by z^k , then the left-hand side can be written in the form $\sum_{0 \leq j \leq k} (1 - z)^j L_{k,j}(\vartheta) Q(z)$ for some polynomials $L_{k,j}$, all of degree k . But this approach does not lead to simpler solutions.

Two special cases are indicative of the general pattern for simplification.

(i) If all q_j 's are zeros, then (2.8) becomes

$$(2.9) \quad (\vartheta^k - 2^k) Q(z) = \frac{2^k - 1}{1 - z}.$$

(ii) If the q_j 's are all ones, then, by the relation

$$\left(\vartheta + \frac{1}{z}\right) \frac{\varphi(z)}{z^j} = \frac{1 - j(1 - z)}{z^{j+1}} \varphi(z) \quad (j = 0, 1, \dots),$$

we have

$$\left(\vartheta - 2^k + \frac{(\vartheta + 1)^k - \vartheta^k}{z}\right) Q(z) = z^{-1} \left(2 - 2^{k+1} + \frac{k2^k}{1 - z}\right),$$

which, by multiplying both sides by z , yields

$$(2.10) \quad ((\vartheta + 1)^k - 2^k) Q(z) - (1 - z) (\vartheta^k - 2^k) Q(z) = 2 - 2^{k+1} + \frac{k2^k}{1 - z}.$$

These observations suggest that we define a “minimum exponent” $\mu_{\mathbf{q}}$ as $k - v + 1$, where v denotes the last position of the first block of 1's (from left to right) and $\mu_{\mathbf{q}} := 0$ if $s = 0$. In symbols,

$$[q_1, \dots, q_k] = [0, \dots, 0, 1, \dots, \underbrace{1, 0, *, \dots, *}_{\mu_{\mathbf{q}}}] \quad (* \in \{0, 1\})$$

so that $\mu_{\mathbf{q}} := 1$ if $s = k$.

LEMMA 2.8. *Let $\mu_{\mathbf{q}}$ be defined as above. Then*

$$(2.11) \quad \begin{aligned} & z^{\mu_{\mathbf{q}}} \left(\left(\vartheta + \frac{q_k}{z}\right) \dots \left(\vartheta + \frac{q_1}{z}\right) - 2^k \right) Q(z) \\ & = P_{k,0}(\vartheta) Q(z) - \sum_{1 \leq \ell \leq \mu_{\mathbf{q}}} (1 - z)^\ell P_{k,\ell}(\vartheta) Q(z), \end{aligned}$$

where the $P_{k,\ell}$'s are polynomials of degree k defined by $P_{k,0}(x) = p_{k,0}(x) - 2^k$, and for $1 \leq \ell \leq \mu_{\mathbf{q}}$,

$$(2.12) \quad P_{k,\ell}(x) := p_{k,\ell}(x) + (-1)^\ell 2^k \binom{\mu_{\mathbf{q}}}{\ell},$$

with

$$\begin{cases} p_{k,1}(x) = (x + q_k - 1)p_{k-1,1}(x) + (x + \mu_{\mathbf{q}} - 1)p_{k-1,0}(x), \\ p_{k,\ell}(x) = (x + q_k - \ell)p_{k-1,\ell}(x) - (x + \mu_{\mathbf{q}} - \ell)p_{k-1,\ell-1}(x) \end{cases} \quad (2 \leq \ell < \mu_{\mathbf{q}}),$$

and the boundary values $p_{k,0}(x) = x^{k-s}(x+1)^s$ and $p_{k,\mu_{\mathbf{q}}}(x) = (-1)^{\mu_{\mathbf{q}}+1}x^k$ for $k \geq 1$.

Proof. The two cases of when $s = 0$ and when $s = k$ are easily checked by (2.9) and (2.10), respectively. We assume that $1 \leq s < k$.

For $k = 2$, there are two query patterns, $[0, 1]$ and $[1, 0]$. The differential equation for $[0, 1]$ is ($\mu_{\mathbf{q}} = 1$)

$$(2.13) \quad (\vartheta(\vartheta + 1) - 4 - (1 - z)(\vartheta^2 - 4)) Q(z) = 1 + \frac{6z}{1 - z},$$

and that for $[1, 0]$ is ($\mu_{\mathbf{q}} = 2$)

$$(2.14) \quad ((\vartheta(\vartheta + 1) - 4) - (1 - z)(2\vartheta^2 + \vartheta - 7) + (1 - z)^2(\vartheta^2 - 4)) Q(z) = \frac{4z^2}{1 - z}.$$

Thus (2.11) holds.

When the query is of the form

$$(2.15) \quad \underbrace{[0, \dots, 0]}_{k-s}, \underbrace{[1, \dots, 1]}_s \quad (1 \leq s < k),$$

we have $\mu_{\mathbf{q}} = 1$, and the left-hand side of (2.8) satisfies

$$z \left(\vartheta + \frac{1}{z} \right)^s \vartheta^{k-s} Q(z) = (\vartheta^{k-s}(\vartheta + 1)^s - 2^k - (1 - z)(\vartheta^k - 2^k)) Q(z);$$

thus (2.11) holds.

The remaining cases follow by induction on k . The proof is messy and omitted here. \square

Similarly, the right-hand side of (2.8) can be rewritten as

$$(2.16) \quad z^{\mu_{\mathbf{q}}} \sum_{1 \leq j \leq k} 2^{j-1} \left(\vartheta + \frac{q_k}{z} \right) \cdots \left(\vartheta + \frac{q_j}{z} \right) \frac{z}{1 - z} = \sum_{-1 \leq \ell < \mu_{\mathbf{q}}} \nu_{k,\ell} (1 - z)^\ell.$$

For the special query pattern (2.15), we have

$$(2.17) \quad \begin{cases} \nu_{k,-1} = (s + 1)2^k - 2^s, \\ \nu_{k,0} = -2^{k+1} + 2^{k-s} + 1. \end{cases}$$

For other cases, we obtain, by induction,

$$(2.18) \quad \nu_{k,\ell} := \begin{cases} (2^{k-1} + \nu_{k-1,-1})(q_k + 1), & \ell = -1, \\ 2^{k-1}(q_k + 1)(-1)^\ell \binom{\mu_{\mathbf{q}}}{\ell + 1} \\ \quad + ((\ell - \mu_{\mathbf{q}})\nu_{k-1,\ell-1} + (q_k - \ell)\nu_{k-1,\ell}), & 0 \leq \ell \leq \mu_{\mathbf{q}} - 2, \\ 2^{k-1}(q_k + 1)(-1)^{\mu_{\mathbf{q}}} - \nu_{k-1,\mu_{\mathbf{q}}-2}, & \ell = \mu_{\mathbf{q}} - 1, \end{cases}$$

with the initial values $\nu_{\tau+1,-1} = 2(2^{\tau+1} - 1)$ and $\nu_{\tau+1,0} = -3 \cdot 2^\tau + 1$, τ being the largest integer for which $q_1 = \dots = q_\tau = 0$ but $q_{\tau+1} = 1$. The proof is again lengthy and laborious, and thus omitted here.

Combining (2.11) and (2.16), we obtain the following “normal form” for Q .

PROPOSITION 2.9. *The generating function $Q(z)$ satisfies the initial value problem*

$$(2.19) \quad \begin{cases} P_{k,0}(\vartheta)Q(z) - \sum_{1 \leq \ell \leq \mu_q} (1-z)^\ell P_{k,\ell}(\vartheta)Q(z) = \sum_{-1 \leq \ell < \mu_q} \nu_{k,\ell}(1-z)^\ell, \\ Q(z) = z + Q_2 z^2 + \dots + Q_{k-1} z^{k-1} + \dots, \end{cases}$$

where the $P_{k,\ell}$'s are polynomials of degree k defined in (2.12) and the $\nu_{k,\ell}$'s are constants given in (2.17) and (2.18).

The exact solution for such a general problem is still not obvious and exists only for a few special cases to be discussed below.

3. Explicitly solvable cases. Exactly solvable cases of (2.19) include $s = 0$, $s = k$, and $k = 2$, where s is the number of specified coordinates.

None specified: $s = 0$. When no coordinate is specified, the search cost is obviously n . The associated differential equation is given by (2.9), with the initial conditions $Q(z) = z + 2z^2 + \dots + (k-1)z^{k-1} + \dots$. The solution is $Q(z) = z/(1-z)^2$ for all $k \geq 1$. This is a case when *the particular solution is itself the exact solution because all initial values coincide.*

All queries specified: $s = k$. In this case, the two Q_n 's on the two sides of the recurrence (2.1) are the same so that Q_n satisfies

$$Q_n = 1 + \frac{2}{n(n+1)} \sum_{1 \leq j < n} (j+1)Q_j \quad (n \geq 1),$$

with $Q_0 = 0$, which is easily solved to be

$$(3.1) \quad Q_n = 2(H_{n+1} - 1) \quad (n \geq 1).$$

The solution is invariant in k . This is nothing but the expected number of comparisons used for an unsuccessful search in random binary search trees; see [6, 23].

Note that $Q(z)$ satisfies the differential equation (2.10), with the initial values $Q(0) = 0$ and

$$Q_j = 1 + [z^{j-2}] \frac{1}{(1-z)^2} \left(1 - \frac{2\Gamma(j+2z)}{\Gamma(j+2)\Gamma(1+2z)} \right) \quad (1 \leq j < k),$$

Γ being the Gamma function. The differential equation (2.10) can be rewritten as

$$\left((\vartheta + 1)^k - 2^k \right) (zQ(z)) = 2 + (k-2)2^k + \frac{k2^k}{1-z} z,$$

which is indeed of Cauchy–Euler type (see [10]). The exact solution is

$$Q(z) = \frac{2}{z(1-z)} \log \left(\frac{1}{1-z} \right) - \frac{2}{1-z},$$

from which one derives (3.1). This is another case *when the particular solution is itself the exact solution.*

2-*d trees*. In this case, $Q[0, 1](z)$ satisfies (2.13) and $Q[1, 0](z)$ satisfies (2.14), both with the initial conditions $Q(0) = 0$ and $Q'(0) = 1$.

Let ${}_2F_1$ denote Gauss's hypergeometric function

$${}_2F_1 \left(\begin{matrix} a, b \\ c \end{matrix} \middle| z \right) = 1 + \sum_{j \geq 1} \frac{a^{\bar{j}} b^{\bar{j}}}{c^{\bar{j}} j!} z^j,$$

where $x^{\bar{j}} = x(x + 1) \cdots (x + j - 1)$ denotes the rising factorial.

LEMMA 3.1 (exact solutions for $Q(z)$). *The generating function of $Q_n[0, 1]$ satisfies*

$$(3.2) \quad Q[0, 1](z) = \frac{13}{2}(1 - z)^{-\alpha} {}_2F_1 \left(\begin{matrix} 2 - \alpha, -2 - \alpha \\ 1 \end{matrix} \middle| z \right) - \frac{7}{2} + 4z - \frac{3}{2}z^2 - \frac{3}{1 - z},$$

and that for $Q_n[1, 0]$

$$(3.3) \quad Q[1, 0](z) = \frac{13}{2}z(1 - z)^{-\alpha} {}_2F_1 \left(\begin{matrix} 3 - \alpha, -1 - \alpha \\ 3 \end{matrix} \middle| z \right) + 2 - \frac{7}{2}z + \frac{3}{2}z^2 - \frac{2}{1 - z},$$

where $\alpha = (\sqrt{17} - 1)/2$.

The corresponding recurrence relations have the forms

$$Q_n[0, 1] = 3 - \frac{2}{n} + \frac{4}{n^2} \sum_{1 \leq j < n} (n - 1 - j)Q_j[0, 1],$$

$$Q_n[1, 0] = 3 - \frac{2}{n} + \frac{4}{n(n + 1)} \sum_{1 \leq j < n} (n - 1 - j + H_{n-1} - H_j)Q_j[1, 0],$$

with $Q_0 = 0$ for both cases.

COROLLARY 3.2 (exact solutions for Q_n). *For $n \geq 3$,*

$$Q_n[0, 1] = \frac{13}{2} \sum_{0 \leq j \leq n} \binom{n - j + \alpha - 1}{n - j} \frac{(2 - \alpha)^{\bar{j}} (-2 - \alpha)^{\bar{j}}}{j! j!} - 3,$$

$$Q_n[1, 0] = 13 \sum_{0 \leq j < n} \binom{n - j + \alpha - 2}{n - 1 - j} \frac{(3 - \alpha)^{\bar{j}} (-1 - \alpha)^{\bar{j}}}{(j + 2)! j!} - 2.$$

Proof. The proof follows from taking coefficients of z^n on both sides of (3.2) and of (3.3). \square

COROLLARY 3.3 (asymptotics of Q_n).

$$Q_n[0, 1] = \frac{13(2\alpha - 3)}{2} \cdot \frac{\Gamma(2\alpha)}{\Gamma(\alpha)^3} n^{\alpha-1} - 3 + O(n^{\alpha-2}),$$

$$Q_n[1, 0] = \frac{13(8 - 5\alpha)}{2} \cdot \frac{\Gamma(2\alpha)}{\Gamma(\alpha)^3} n^{\alpha-1} - 2 + O(n^{\alpha-2}).$$

Proof. The proof follows from applying the exact solutions of Q_n or by applying singularity analysis to (3.2) and (3.3), and then using Gauss's identity (see [1, Eq. 15.1.20, p. 556]). \square

Numerically,

$$\frac{13(2\alpha - 3)}{2} \cdot \frac{\Gamma(2\alpha)}{\Gamma(\alpha)^3} \approx 2.55275, \quad \frac{13(8 - 5\alpha)}{2} \cdot \frac{\Gamma(2\alpha)}{\Gamma(\alpha)^3} \approx 1.99312,$$

which are to be compared with the asymptotic approximation

$$\frac{1}{2} \cdot \frac{\Gamma(2\alpha)}{\Gamma(\alpha)^3} n^{\alpha-1} \approx 1.59099 n^{\alpha-1}$$

for the expected cost of partial match queries in random two-dimensional quadrees; see [9, 14].

Proof of Lemma 3.1. Consider first the case when the query pattern is $[0, 1]$. In this case, the differential equation is

$$z(1-z)^2 Q''(z) + (1-z)^2 Q'(z) - 4zQ(z) = -5 + \frac{6}{1-z}.$$

To solve this equation, we first observe that the particular solution is given by

$$y_p(z) = -\frac{7}{2} + 4z - \frac{3}{2}z^2 - \frac{3}{1-z}.$$

Thus we consider $Q(z) = (1-z)^{-\alpha}y(z) + y_p(z)$, and we have

$$z(1-z)^2 y''(z) + (1-z)(1-z+2z\alpha)y'(z) + (\alpha+z\alpha^2-4z)y(z) = 0.$$

Since $\alpha(\alpha+1) - 4 = 0$, we can simplify the above equation and obtain

$$z(1-z)y''(z) + (1+(2\alpha-1)z)y'(z) + \alpha y(z) = 0.$$

This equation is then rewritten as a generalized hypergeometric differential equation

$$(\delta^2 - z(\delta^2 - 2\alpha\delta - \alpha))y(z) = 0,$$

where $\delta = z(d/dz)$. The differential equation has two fundamental solutions, and the only one that is regular at the origin is

$${}_2F_1\left(\begin{matrix} 2-\alpha, -2-\alpha \\ 1 \end{matrix} \middle| z\right).$$

Thus the solution is of the form

$$Q(z) = -\frac{7}{2} + 4z - \frac{3}{2}z^2 - 3(1-z)^{-1} + c_0(1-z)^{-\alpha}{}_2F_1\left(\begin{matrix} 2-\alpha, -2-\alpha \\ 1 \end{matrix} \middle| z\right),$$

and by matching the initial values, we conclude (3.2).

For the query pattern $[1, 0]$, the particular solution for (2.14) is

$$y_p(z) := 2 - \frac{7}{2}z + \frac{3}{2}z^2 - \frac{2}{1-z}.$$

The only different part of the above proof is considering $y(z) := (Q[1, 0](z) - y_p(z)) / z$; the remaining analysis is similar and omitted. \square

The approach we used for deriving (3.2) and (3.2) is similar to the one used in [14]. However, such an approach does not apply for $k \geq 3$. We need a different approach to solving (2.19).

4. Constants for general query patterns. We rewrite the differential equation (2.19) as

$$(4.1) \quad P_0(\vartheta)Q(z) = \sum_{1 \leq \ell \leq \mu} (1-z)^\ell P_\ell(\vartheta)Q(z) + \sum_{-1 \leq \ell < \mu} \nu_{k,\ell}(1-z)^\ell,$$

where, for simplicity, we write $P_\ell = P_{k,\ell}$ (defined in (2.12)) and $\mu = \mu_{\mathbf{q}}$.

Our main result in this paper is the following asymptotic estimate for Q_n .

THEOREM 4.1. *The expected cost of a random partial match query in a random k - d tree of n nodes satisfies*

$$(4.2) \quad Q_n \sim \frac{K}{\Gamma(\alpha)} n^{\alpha-1} \quad (1 \leq s < k),$$

where the constant K is defined in (4.6) below.

This theorem is proved in the next section by extending the asymptotic theory in [10] for Cauchy–Euler differential equations. The idea is roughly as follows. If the right-hand side of (4.1) is independent of Q , then the solution of Q would satisfy $Q(z) \sim c_1(1-z)^{-\alpha} + y_p(z)$ for some constant c_1 and particular solution $y_p(z)$; see [10]. But the intricate part here is that the right-hand side of (4.1), expected to have a smaller contribution to $Q(z)$, itself depends on Q .

In this section we give an approach to the proof of (4.2) based on Mellin-type integrals. The proof is almost complete, up to an estimate in the complex plane that is needed in justifying the Mellin inversion integrals. The advantage of this approach is that it quickly gives the right form for the constant.

Shifting the initial values. A minor but crucial step in our analysis is to consider the function $f(z) := Q(z) - \sum_{1 \leq j < k} Q_j z^j$. The differential equation for f remains the same, but all initial values become zero. Indeed, by the integral equation (2.7), we have

$$f(z) - 2^k(I_{q_1} \circ \dots \circ I_{q_k})[f](z) = G_1(z),$$

with $f(0) = \dots = f^{(k-1)}(0) = 0$, where

$$G_1(z) := G_0(z) - \sum_{1 \leq j < k} Q_j z^j - 2^k(I_{q_1} \circ \dots \circ I_{q_k}) \left[\sum_{1 \leq j < k} Q_j z^j \right] (z).$$

Note that $I_q[z^j] = z^{j+1}/(j+1+q) + \dots$, $q \in \{0, 1\}$. This, together with the relations $Q_j = [z^j]G_0(z)$ for $0 \leq j < k$, implies that $[z^j]G_1(z) = 0$ for $j < k$.

Now applying the inverse operators to the above integral equation, and then multiplying both sides by z^μ , we obtain

$$(4.3) \quad P_0(\vartheta)f(z) = \sum_{1 \leq \ell \leq \mu} (1-z)^\ell P_\ell(\vartheta)f(z) + g(z),$$

with $f^{(j)}(0) = 0$, $0 \leq j \leq k-1$, where

$$g(z) = z^\mu \sum_{1 \leq j \leq k} 2^{j-1} \left(\vartheta + \frac{q_k}{z} \right) \dots \left(\vartheta + \frac{q_j}{z} \right) \frac{z}{1-z} - z^\mu \left(\left(\vartheta + \frac{q_k}{z} \right) \dots \left(\vartheta + \frac{q_1}{z} \right) - 2^k \right) (Q_1 z + \dots + Q_{k-1} z^{k-1}).$$

By using the same arguments as in the proof of Proposition 2.9, we obtain the alternative form for g ,

$$g(z) = \frac{\nu_{k,-1}}{1-z} + \Pi(z),$$

with $\Pi(z)$ being a polynomial of degree $k + \mu - 1$. With this form, we can now define

$$(4.4) \quad g^*(w) := \int_0^1 (1-x)^{w-1} g(x) dx \quad (\Re(w) > 0).$$

Then g^* satisfies

$$g^*(w) = \frac{\nu_{k,-1}}{w-1} + \sum_{0 \leq j < k+\mu} \frac{(-1)^j \Pi^{(j)}(1)}{j!(w+j)};$$

see Table 5.2. The function g^* can be viewed either as the Mellin transform (see [15]) of $g(x)$ (defined to be zero for $x > 1$) or as the factorial series of the sequence $[z^j]g(z)$.

The following estimate is needed in proving the absolute convergence of the series representation (4.6) of K .

LEMMA 4.2. *The function g^* satisfies the estimate*

$$(4.5) \quad g^*(w) = O(|w|^{-\mu-1})$$

for large $|w|$ and $|\arg(w)| \leq \pi - \varepsilon$, $\varepsilon > 0$.

Proof. Observe first that by definition,

$$g(z) = z^\mu \left(\vartheta + \frac{q_k}{z}\right) \cdots \left(\vartheta + \frac{q_j}{z}\right) G_1(z),$$

and that $[z^j]G_1(z) = 0$ for $j < k$. Since each operator $\vartheta + q_k/z$ has the effect of decreasing the powers of monomials by 1 (from z^j to z^{j-1}), we can expand g as $g(z) = \sum_{j \geq \mu} g_j z^j$ for some coefficients g_j . By using this expansion and interchanging the summation and integral, we obtain

$$g^*(w) = \sum_{j \geq \mu} \frac{g_j j!}{w(w+1) \cdots (w+j)} \quad (\Re(w) > 1),$$

which is not only a factorial series but also an asymptotic expansion for large $|w|$. □

Characterization of K . Now we are ready to give an explicit form for the constant K in Theorem 4.1.

PROPOSITION 4.3. *The constant K in (4.2) is given by*

$$(4.6) \quad K = \frac{1}{P'_0(\alpha)} \sum_{j \geq 0} g^*(\alpha + j) B_j,$$

with the series being absolutely convergent, where

$$(4.7) \quad B_j = \sum_{1 \leq \ell \leq \mu} \frac{P_\ell(\alpha + j)}{P_0(\alpha + j)} B_{j-\ell} \quad (j \geq 1),$$

with the initial conditions $B_0 = 1$ and $B_j = 0$ for $j < 0$.

We defer the hard part of proving (4.6) to later sections and prove here only the absolute convergence, which is a direct consequence of the estimate (4.5) for g^* and the following lemma.

LEMMA 4.4. *Let $B(z) = \sum_{j \geq 0} B_j z^j$. Then $B(z)$ satisfies the differential equation*

$$(4.8) \quad (1 - z)^\mu \left(\left(\frac{q_1 - \mu z}{1 - z} + \alpha + z \mathbb{D}_z \right) \cdots \left(\frac{q_k - \mu z}{1 - z} + \alpha + z \mathbb{D}_z \right) - 2^k \right) B(z) = 0,$$

where $\mathbb{D}_z := d/dz$, and the sequence B_j satisfies

$$(4.9) \quad B_j = O(j^{\mu - q_k - 1}).$$

Proof. The proof is adapted from that of Lemma 3 in [7]. Define the operator

$$\Psi(q_1, \dots, q_k)[\vartheta] := \left(\vartheta + \frac{q_k}{z} \right) \cdots \left(\vartheta + \frac{q_1}{z} \right).$$

By the Cauchy integral representation,

$$B_j = \frac{1}{2\pi i} \oint (1 - z)^{-j-1} B(1 - z) dz \quad (j \geq 0),$$

where, here and throughout the proof, the integration contour is a sufficiently small circle around unity, we obtain

(4.10)

$$\begin{aligned} 0 &= \frac{1}{2\pi i} \oint B(1 - z)(1 - z)^{\alpha-1} \\ &\quad \times \left(P_0(j + \alpha) - \sum_{1 \leq \ell \leq \mu} (1 - z)^\ell P_\ell(j + \alpha) \right) (1 - z)^{-j-\alpha} dz \\ &= \frac{1}{2\pi i} \oint B(1 - z)(1 - z)^{\alpha-1} \left(P_0(\vartheta) - \sum_{1 \leq \ell \leq \mu} (1 - z)^\ell P_\ell(\vartheta) \right) (1 - z)^{-j-\alpha} dz \\ &= \frac{1}{2\pi i} \oint B(1 - z)(1 - z)^{\alpha-1} (z^\mu \Psi(q_1, \dots, q_k)[\vartheta] - 2^k z^\mu) (1 - z)^{-j-\alpha} dz \\ &= V_j - 2^k [z^j] B(z)(1 - z)^\mu, \end{aligned}$$

where

$$\begin{aligned} V_j &:= \frac{1}{2\pi i} \oint B(1 - z)(1 - z)^{\alpha-1} z^\mu \Psi(q_1, \dots, q_k)[\vartheta] (1 - z)^{-j-\alpha} dz \\ &= \frac{1}{2\pi i} \oint B(1 - z)(1 - z)^{\alpha-1} z^\mu \left(\vartheta + \frac{q_k}{z} \right) \Psi(q_1, \dots, q_{k-1})[\vartheta] (1 - z)^{-j-\alpha} dz. \end{aligned}$$

By an integration by parts (for the term corresponding to ϑ), we have

$$V_j = \frac{1}{2\pi i} \oint B_1(1 - z)(1 - z)^{\alpha-1} z^\mu \Psi(q_1, \dots, q_{k-1})[\vartheta] (1 - z)^{-j-\alpha} dz,$$

where

$$B_1(1 - z) := \left(\frac{q_k - \mu(1 - z)}{z} + \alpha - \vartheta \right) B(1 - z).$$

Repeating the same argument $k - 1$ times, we obtain

$$\begin{aligned} V_j &= \frac{1}{2\pi i} \oint (1 - z)^{-j-1} z^\mu \\ &\quad \times \left(\frac{q_1 - \mu(1 - z)}{z} + \alpha - \vartheta \right) \cdots \left(\frac{q_k - \mu(1 - z)}{z} + \alpha - \vartheta \right) B(1 - z) dz \\ &= [z^j] (1 - z)^\mu \left(\frac{q_1 - \mu z}{1 - z} + \alpha + z\mathbb{D}_z \right) \cdots \left(\frac{q_k - \mu z}{1 - z} + \alpha + z\mathbb{D}_z \right) B(z). \end{aligned}$$

From this and (4.10), the result (4.8) follows from multiplying both sides of (4.10) and summing over all $j \geq 0$.

According to the Frobenius method (see [24]), we seek solutions to the differential equation (4.8) of the form $B(z) = (1 - z)^\zeta \xi(1 - z)$ for some $\zeta \in \mathbb{C}$ and some function $\xi(z)$ analytic at the origin. Substituting this form into (4.8), we see that the indicial equation for (4.8) is given by

$$\prod_{1 \leq j \leq k} (-\mu - \zeta + k - j + q_j) = 0.$$

In particular, the dominant zero is $\zeta = -\mu + q_k$, which is a simple zero. This implies that

$$B(z) = O(|1 - z|^{-\mu + q_k}),$$

and, by singularity analysis (see [16]), proves (4.9). \square

Note that, by (4.7), $B(z)$ also satisfies the differential equation

$$P_0(\alpha + z\mathbb{D}_z)B(z) = \sum_{1 \leq \ell \leq \mu} (1 - z)^\ell P_\ell(\alpha + \ell + z\mathbb{D}_z)B(z),$$

but this form is less manageable than (4.8) for our purposes.

While the series form (4.6) may seem recursive, it is readily modified for numerical purposes; see Table 5.1.

We next give a formal proof of the formula (4.6) by Mellin integrals, which will be justified by a more algebraic procedure.

Factorial series and Mellin integrals. Define the factorial series

$$f^*(w) := \int_0^1 (1 - z)^{w-1} f(z) dz = \sum_{j \geq k} \frac{Q_j j!}{w(w + 1) \cdots (w + j)}.$$

Then $f^*(w)$ is well defined in the half-plane $\Re(w) > \alpha$.

By a Mellin inversion formula (or by a standard argument for the integral representation for factorial series), we have

$$(4.11) \quad f(z) = \frac{1}{2\pi i} \int_{\sigma - i\infty}^{\sigma + i\infty} f^*(w) (1 - z)^{-w} dw \quad (\sigma > \alpha).$$

Substituting this into (4.3) and using (4.4), we obtain

$$\begin{aligned} &\frac{1}{2\pi i} \int_{\sigma - i\infty}^{\sigma + i\infty} f^*(w) P_0(w) (1 - z)^{-w} dw \\ &= \frac{1}{2\pi i} \int_{\sigma - i\infty}^{\sigma + i\infty} f^*(w) \sum_{1 \leq \ell \leq \mu} P_\ell(w) (1 - z)^{-w + \ell} dw + \frac{1}{2\pi i} \int_{\sigma - i\infty}^{\sigma + i\infty} g^*(w) (1 - z)^{-w} dw. \end{aligned}$$

By absolute convergence and by the changes of variables $w \mapsto w + j$, we are led to the difference equation

$$(4.12) \quad f^*(w) = \sum_{1 \leq \ell \leq \mu} \frac{P_\ell(w + \ell)}{P_0(w)} f^*(w + \ell) + \frac{g^*(w)}{P_0(w)}$$

for $\Re(w) > \alpha$, with the additional property that $f^*(w) \rightarrow 0$ as $|w| \rightarrow \infty$ in the half-plane $|\arg(w)| < \pi$. But the right-hand side of (4.12) also gives a meromorphic continuation of $f^*(w)$ to the whole plane (up to the zeros of $P_0(w + \ell)$ for $\ell \geq 0$).

In particular, since α is a simple zero of $P_0(w)$ (see [9]),

$$(4.13) \quad f^*(w) \sim \frac{K'}{w - \alpha} \quad (w \sim \alpha),$$

where the residue K' can be computed by

$$(4.14) \quad \begin{aligned} K' &:= \lim_{w \rightarrow \alpha} (w - \alpha) f^*(w) \\ &= \frac{1}{P'_0(\alpha)} \left(\sum_{1 \leq \ell \leq \mu} P_\ell(\alpha + \ell) f^*(\alpha + \ell) + g^*(\alpha) \right). \end{aligned}$$

Mellin inversion and singularity analysis. Heuristically, the series form (4.6) results easily from (4.13) and the Mellin inversion formula (4.11) as follows. First, by (4.11) using (4.13), we expect that

$$f(z) \sim K'(1 - z)^{-\alpha},$$

by formally shifting the integration line to $\Re(w) = \alpha - \varepsilon$ and by taking into account the residue of the integrand at $w = \alpha$.

Then by a formal application of the singularity analysis, we anticipate the approximation

$$[z^n]Q(z) = [z^n]f(z) \sim \frac{K'}{\Gamma(\alpha)} n^{\alpha-1},$$

so that K' should equal K .

Justifying the above quick analysis requires an estimate of $|f^*(\sigma \pm iT)|$ for large T . On the other hand, because of the presence of the factor $(1 - z)^{-w}$, which can be exponentially large when $z \in \mathbb{C}$ lies near unity and when T grows, we need an estimate of $|f^*(\sigma \pm iT)|$ that decays at infinity at an exponential rate (in T). While this approach might be made rigorous, we prefer to develop another approach that is more general and does not rely on analytic properties but instead on algebraic manipulations of integrals.

Effective expressions for K . Iterating the right-hand side of (4.14) N times by the same difference equation (4.12), we deduce that

$$(4.15) \quad P'_0(\alpha)K = \sum_{1 \leq j \leq (\mu-1)N+1} A_{N,j} f^*(\alpha + N + j - 1) + K_N$$

for any $N \geq 1$, where $A_{1,j} := P_j(\alpha + j)$ for $1 \leq j \leq \mu$ and for $N \geq 2$,

$$(4.16) \quad A_{N,\ell} = \sum_{1 \leq i \leq \mu} \frac{P_i(\alpha + N + \ell - 1)}{P_0(\alpha + N + \ell - 1 - i)} A_{N-1,\ell+1-i} \quad (1 \leq \ell \leq N(\mu - 1) + 1),$$

and

$$\begin{aligned} K_N &= K_{N-1} + \sum_{1 \leq j \leq (N-1)(\mu-1)+1} \frac{g^*(\alpha + N + j - 2)}{P_0(\alpha + N + j - 2)} A_{N-1,j} \\ &= g^*(\alpha) + \sum_{1 \leq i \leq N-1} \sum_{1 \leq j \leq i(\mu-1)+1} \frac{g^*(\alpha + i + j - 1)}{P_0(\alpha + i + j - 1)} A_{i,j} \\ &= g^*(\alpha) + \sum_{1 \leq j \leq (N-1)\mu} \frac{g^*(\alpha + j)}{P_0(\alpha + j)} \sum_{1 \leq i \leq j} A_{i,j+1-i}. \end{aligned}$$

By induction and the recursive expression for P_j 's, we have $P_j(x) \asymp x^k$ for large x . From this and the estimate (4.5), we obtain

$$K = \frac{1}{P'_0(\alpha)} \lim_{N \rightarrow \infty} K_N = \frac{1}{P'_0(\alpha)} \left(g^*(\alpha) + \sum_{j \geq 1} \frac{g^*(\alpha + j)}{P_0(\alpha + j)} \sum_{1 \leq i \leq j} A_{i,j+1-i} \right).$$

Define

$$B_j := \frac{1}{P_0(\alpha + j)} \sum_{1 \leq i \leq j} A_{i,j+1-i}.$$

Then B_j is easily seen to satisfy (4.7) by using (4.16).

5. A Cauchy–Euler approach. In this section we develop a different approach to proving (4.6). The reason we write (4.3) in the form of a Cauchy–Euler differential equation is that if

$$(5.1) \quad f(z) = (1 - z)^{-\alpha} \xi(1 - z)$$

for $z \in \mathbb{C}$ near unity, where $\xi(z)$ is analytic at the origin, then

$$(5.2) \quad \sum_{1 \leq \ell \leq \mu} (1 - z)^\ell P_\ell(\vartheta) f(z) = O(|1 - z|^{-\alpha+1}),$$

and thus the dominant (asymptotic) solutions are expected to be determined by the left-hand side of (4.3), with the right-hand side behaving as if it is independent of f . We apply again the Frobenius method to prove (5.1). Once (5.1) is proved, (4.3) is then solved asymptotically by extending our approach of *iterative linear operators* developed in [10].

The growth order of f . By using the formula

$$\vartheta^j f(z) = \sum_{0 \leq \ell \leq j} (-1)^{j+\ell} S(j, \ell) (1 - z)^\ell f^{(\ell)}(z),$$

where the $S(j, \ell)$'s represent the Stirling numbers of the second kind, we can rewrite (4.3) in the form

$$\sum_{0 \leq j \leq k} L_j(z) (1 - z)^j f^{(j)}(z) = g(z)$$

for some polynomials L_j of degree k . In particular, $L_k(z) = z^\mu (1 - z)^k$. Therefore, the possible singularities of (4.3) are the two zeros $z = 0$ and $z = 1$ of the polynomial

$z^\mu(1 - z)^k$, $z = \infty$, and the singularities of $g(z)$ ($z = 1$ only). In particular, the singularity $z = 1$ is a *regular singular point*.

Before applying the Frobenius method (see [24]), we use suitable operators to convert the nonhomogeneous equation into a homogeneous one. We start from the relation

$$(\vartheta + j)(1 - z)^j = 0 \quad (j \in \mathbb{Z}).$$

Then we can annihilate the nonhomogeneous term in (4.3) by multiplying both sides by a sufficient number of the above annihilating operators, giving

$$(5.3) \quad \left(\prod_{-1 \leq j < k + \mu} (\vartheta + j) \right) \left(P_0(\vartheta)f(z) - \sum_{1 \leq \ell \leq \mu} (1 - z)^\ell P_\ell(\vartheta)f(z) \right) = 0.$$

This means that if $f(z)$ solves the nonhomogeneous equation (4.3), then it is also a solution of (5.3) since the dominant growth order of f near the singularity $z = 1$ is determined by the zero of the new indicial equation

$$\left(\prod_{-1 \leq j \leq d} (\vartheta + j) \right) P_0(\vartheta)$$

with the largest real part. By the Frobenius method (see [24]), we then deduce the required estimate (5.1), from which (5.2) follows.

Method of iterative linear operators. As in [10], let α_j , $1 \leq j \leq k$, be the zeros of $P_0(z)$ arranged in decreasing order of their real parts (see [9])

$$\alpha = \alpha_1 > \Re(\alpha_2) \geq \dots \geq \Re(\alpha_k).$$

Define

$$\mathbf{I}_\eta[f](z) = (1 - z)^{-\eta} \int_0^z (1 - x)^{\eta-1} f(x) dx.$$

To solve (4.3), we first factor $P_0(\vartheta)$ into linear operators

$$P_0(\vartheta)f(z) = (\vartheta - \alpha_1) \cdots (\vartheta - \alpha_k)f(z) = R(z),$$

where

$$R(z) := g(z) + \sum_{1 \leq j \leq \mu} (1 - z)^j P_j(\vartheta)f(z),$$

and then solve the linear equations one after another, giving (see [10])

$$f(z) = (\mathbf{I}_{\alpha_k} \circ \mathbf{I}_{\alpha_{k-1}} \circ \dots \circ \mathbf{I}_\alpha)[R](z).$$

By using successive integrations by parts (see [8]) or the inductive arguments used in [10], we deduce that

$$f(z) = \frac{1}{P'_0(\alpha)} \mathbf{I}_\alpha[R](z) + T_1(z),$$

where $[z^n]T_1(z) = o(n^{\alpha-1})$.
 Then (see [10])

$$\begin{aligned} Q_n &= [z^n]f(z) = \frac{1}{P'_0(\alpha)}[z^n]\mathbf{I}_\alpha[R](z) + o(n^{\alpha-1}) \\ &= \frac{1}{P'_0(\alpha)} \sum_{0 \leq \ell < n} \frac{[z^\ell]R(z)}{\ell + 1} \prod_{\ell+2 \leq j \leq n} \left(1 + \frac{\alpha - 1}{j}\right) + o(n^{\alpha-1}) \\ &= \frac{K''}{P'_0(\alpha)\Gamma(\alpha)} n^{\alpha-1} + o(n^{\alpha-1}), \end{aligned}$$

where

$$K'' = \int_0^1 (1-x)^{\alpha-1} R(x) dx = \sum_{\ell \geq 0} \frac{\ell! [z^\ell]R(z)}{\alpha(\alpha+1) \cdots (\alpha+\ell)}.$$

Iteration of the leading constant. We next “mimic” the operations of the Mellin approach and show that

$$K'' = P'_0(\alpha)K,$$

and this will complete the proof of Theorem 4.1.

LEMMA 5.1. *Let $\omega(x)$ and $\beta(x)$ be two polynomials of degree at most k . Let $h(x)$ be defined in the unit interval. If h satisfies*

- (i) $h^{(j)}(0) = 0$ for $0 \leq j < k$, and
- (ii) *the integral*

$$h^*(w) := \int_0^1 (1-x)^{\eta-1} h(x) dx$$

converges,

then for any $\eta \in \mathbb{C}$ for which $\beta(\eta) \neq 0$,

$$(5.4) \quad \int_0^1 (1-x)^{\eta-1} \omega(\vartheta_x) \beta(\vartheta_x)^{-1} h(x) dx = \frac{\omega(\eta)}{\beta(\eta)} h^*(\eta),$$

where $\beta(\vartheta_x)^{-1}$ represents the inverse operator of the differential operator $\beta(\vartheta_x)$ and $\vartheta_x := (1-x)(d/dx)$.

Proof. Obviously, (5.4) holds when $\omega(x) = \beta(x)$. It is also easily checked, by integration by parts, for the two fundamental cases $\omega(x) = x$, $\beta(x) = 1$ and $\omega(x) = 1$, $\beta(x) = x - \nu$ for $\nu \neq \eta$.

When $\omega(x)$ is a polynomial of degree at most k , we first write $\omega(x) = (x-\nu)\omega_1(x)$, where ν is a zero of $\omega(x)$. Then

$$\begin{aligned} \int_0^1 (1-x)^{\eta-1} \omega(\vartheta_x) h(x) dx &= \int_0^1 (1-x)^{\eta-1} (\vartheta_x - \nu) \omega_1(\vartheta_x) h(x) dx \\ &= (\eta - \nu) \omega_1(\vartheta_x) h^*(\eta). \end{aligned}$$

Repeating the same argument, we have

$$\int_0^1 (1-x)^{\eta-1} \omega(\vartheta_x) h(x) dx = \omega(\eta) h^*(\eta)$$

and

$$\int_0^1 (1-x)^{\eta-1} \beta(\vartheta_x)^{-1} h(x) dx = \frac{h^*(\eta)}{\beta(\eta)}.$$

Similarly, we derive (5.4). \square

Define

$$\Lambda_f(\alpha) := \int_0^1 (1-x)^{\alpha-1} P_0(\vartheta_x) f(x) dx.$$

Note that $\Lambda_f(\alpha) = K''$. By substituting (4.3) into the above integral and by applying (5.4), we have

$$(5.5) \quad \Lambda_f(\alpha) = g^*(\alpha) + \sum_{1 \leq \ell \leq \mu} \frac{P_\ell(\alpha + \ell)}{P_0(\alpha + \ell)} \Lambda_f(\alpha + \ell),$$

where $g^*(\alpha)$ is defined as in (4.4). By definition, we have

$$K'' = g^*(\alpha) + \sum_{1 \leq \ell \leq \mu} \frac{P_\ell(\alpha + \ell)}{P_0(\alpha + \ell)} \Lambda_f(\alpha + j).$$

It follows by (4.12) and the property $\Lambda_f(x) \rightarrow 0$ as $x \rightarrow \infty$ that $K'' = P'_0(\alpha)K$, as required.

More refined approximations. We can refine the above analysis and derive the effective approximation

$$(5.6) \quad Q_n = \frac{K}{\Gamma(\alpha)} n^{\alpha-1} - \frac{1}{2^k - 2^s} \sum_{1 \leq j \leq k} 2^{j-1+q_j+\dots+q_k} + O\left(n^{\alpha-2} + n^{\Re(\alpha_2)-1} (\log n)^{\kappa-1}\right),$$

where κ denotes the largest multiplicity of zeros with real parts equal to $\Re(\alpha_2)$ (see [10]).

Hypergeometric cases: $\mu = 1$. When the query pattern is of the form (2.15), we can obtain more explicit expressions. In this case, $\mu = 1$ and $P_1(x) = x^k - 2^k$. Then

$$K = \frac{1}{P'_0(\alpha)} \sum_{j \geq 0} g^*(\alpha + j) \prod_{1 \leq \ell \leq j} \frac{P_1(\alpha + \ell)}{P_0(\alpha + \ell)},$$

where, by a lengthy calculation,

$$g^*(w) := \frac{(s+1)2^k - 2^s}{w-1} + \frac{2^k \sigma_0 - 2^{k+1} + 2^{k-s} + 1}{w} - \sum_{1 \leq \ell \leq k} \frac{P_0(-\ell) \sigma_\ell - P_1(1-\ell) \sigma_{\ell-1}}{w+\ell},$$

with $\sigma_\ell := (-1)^\ell \sum_{\ell \leq j \leq k} \binom{j}{\ell} Q_j$. From this expression of g , we can further rewrite the above series form for K in terms of generalized hypergeometric functions.

$\mu \geq 2$. When $\mu \geq 2$, the solution to the recurrence (4.7) is, in general, less explicit. But there are special cases when $\mu > 1$ can be reduced to $\mu = 1$. These occur when the query patterns are of the form

$$\underbrace{[0, \dots, 0]}_{k-\mu-s+1}, \underbrace{[1, \dots, 1]}_s, [0, \dots, 0] \quad (1 \leq s \leq k - \mu + 1),$$

TABLE 5.1

Approximate numerical values of α and $Q_n/n^{\alpha-1}$ for all generating query patterns (up to cyclic rotations) for $k = 3, 4, 5$, and 6. Note that the periodic patterns $[0, 1, 0, 1]$, $[0, 1, 0, 1, 0, 1]$, $[0, 0, 1, 0, 0, 1]$ and $[0, 1, 1, 0, 1, 1]$ are not shown. The total number of different patterns (up to rotations) needed to cover all 2^k possible query forms is given by (2.4).

k	Query	$\alpha \approx$	$Q_n/n^{\alpha-1} \approx$
3	$[0, 0, 1]$	1.71618 86589 93105	1.88700 34494 16788
	$[0, 1, 1]$	1.39485 86738 66065	4.30626 60684 05608
4	$[0, 0, 0, 1]$	1.78995 09772 69481	1.66842 10542 80183
	$[0, 0, 1, 1]$	1.56155 28128 08830	2.91912 77264 05377
	$[0, 1, 1, 1]$	1.30555 31614 36616	6.16148 43696 18751
5	$[0, 0, 0, 0, 1]$	1.83323 02942 30338	1.55981 70883 35171
	$[0, 0, 0, 1, 1]$	1.65556 26632 48591	2.46232 74709 13660
	$[0, 0, 1, 0, 1]$	1.65556 26632 48591	2.21247 44030 89626
	$[0, 0, 1, 1, 1]$	1.46323 88095 76994	4.05206 48849 27873
	$[0, 1, 1, 0, 1]$	1.46323 88095 76994	3.24485 54614 39396
	$[0, 1, 1, 1, 1]$	1.24956 22677 97953	8.07657 98073 98937
6	$[0, 0, 0, 0, 0, 1]$	1.86170 55962 67907	1.49483 95666 52432
	$[0, 0, 0, 0, 1, 1]$	1.71618 86589 93105	2.23506 40352 34654
	$[0, 0, 0, 1, 0, 1]$	1.71618 86589 93105	2.04449 79159 05194
	$[0, 0, 0, 1, 1, 1]$	1.56155 28128 08830	3.35792 47385 71823
	$[0, 0, 1, 1, 0, 1]$	1.56155 28128 08830	2.79761 13698 36274
	$[0, 0, 1, 0, 1, 1]$	1.56155 28128 08830	3.03451 26969 10640
	$[0, 1, 1, 1, 0, 1]$	1.39485 86738 66065	3.95385 24193 61293
	$[0, 0, 1, 1, 1, 1]$	1.39485 86738 66065	5.25849 03652 85129
	$[0, 1, 1, 1, 1, 1]$	1.21106 87077 39977	10.0301 95663 53780

TABLE 5.2

Coefficients of $(w + j)^{-1}$, $-1 \leq j < k + \mu$ in $g^*(w)$ for $k \leq 5$.

Query	-1	0	1	2	3	4	5	6	7
$[0, 0, 1]$	14	13	-64	69	-32				
$[0, 1, 1]$	20	11	-64	65	-32				
$[0, 0, 0, 1]$	30	73	-320	298	114	-195			
$[0, 0, 1, 1]$	44	69	-320	342	60	-195			
$[0, 1, 1, 1]$	56	$\frac{571}{9}$	$-\frac{2752}{9}$	$\frac{1034}{3}$	$\frac{200}{9}$	$-\frac{1625}{9}$			
$[0, 0, 0, 0, 1]$	62	273	-1280	2670	-5926	8425	-4224		
$[0, 0, 0, 1, 1]$	92	265	-1280	2390	-4900	7657	-4224		
$[0, 0, 1, 0, 1]$	76	$\frac{994}{9}$	$-\frac{15400}{9}$	$\frac{46513}{9}$	$-\frac{33088}{3}$	$\frac{180400}{9}$	$-\frac{219772}{9}$	$\frac{142805}{9}$	$-\frac{12320}{3}$
$[0, 0, 1, 1, 1]$	120	$\frac{2317}{9}$	$-\frac{11360}{9}$	$\frac{6634}{3}$	$-\frac{37144}{9}$	$\frac{62165}{9}$	$-\frac{12320}{3}$		
$[0, 1, 1, 0, 1]$	88	$\frac{524}{9}$	$-\frac{4616}{3}$	$\frac{13757}{3}$	$-\frac{82924}{9}$	$\frac{49202}{3}$	$-\frac{61148}{3}$	$\frac{122495}{9}$	-3608
$[0, 1, 1, 1, 1]$	144	$\frac{2099}{9}$	$-\frac{10424}{9}$	$\frac{5818}{3}$	$-\frac{29600}{9}$	$\frac{51647}{9}$	-3608		

the expected cost of which can be computed via that of the special patterns (2.15) and the relations (2.3).

Tables. For concreteness, we give numerical approximations to $K/\Gamma(\alpha)$ in Table 5.1 and the expansions of $g^*(w)$ for $k \leq 5$ in Table 5.2.

Note that the general terms in (4.6) converge slowly; thus it is more efficient, for numerical purposes, to use the expression (4.15); see Table 5.1. The idea is roughly to take a sufficiently large N , say 1000, to compute the first N terms as precisely as we can, and then to estimate the errors by the values of $f^*(\alpha + N)$, which can be easily computed by

$$f^*(\alpha + N) = \sum_{j \geq k} \frac{Q_j j!}{(\alpha + N) \cdots (\alpha + N + j)},$$

the series itself being an asymptotic expansion.

6. k -d- t trees. In this section we extend our approach to k -d- t trees, $t \in \mathbb{N}$, which are locally balanced versions of k -d trees (in which all trees of sizes larger than $2t$ have both subtrees of sizes at least t); see [11]. For simplicity, trees of sizes $\leq 2t$ are not rearranged.

The probabilistic model for partial match queries remains the same as above. For convenience, we use the same set of notations as for k -d trees (indexed by t when ambiguity may arise). Let $Q_{n,t} = Q_n[q_1, \dots, q_k; t]$ stand for the expected number of nodes visited when performing the range search algorithm of a random partial match query in a random k -d- t tree of n nodes. Then $Q_{n,0} = Q_n$. Cunto, Lau, and Flajolet [11] showed that

$$Q_{n,t} \sim C_t n^{\alpha-1}$$

for some constant C_t , where $\alpha > 1$ solves the equation $P_0(\alpha) = 0$, with

$$(6.1) \quad P_0(x) := \left((x+t)^{\overline{t+1}} \right)^{k-s} \left((x+t+1)^{\overline{t+1}} \right)^s - \left((t+2)^{\overline{t+1}} \right)^k.$$

We show that C_t can be computed as above but with more complicated components.

LEMMA 6.1 (basic recurrences). *Define $\varpi_{n,j} := \binom{j}{t} \binom{n-1-j}{t} / \binom{n}{2t+1}$. The expected search cost $Q_{n,t}$ for fixed $t \in \mathbb{N}$ satisfies the system of recurrences*

$$(6.2) \quad \begin{cases} Q_n[q_1, \dots, q_k; t] = 1 + 2 \sum_{1 \leq j < n} \rho_{n,j}(q_1) \varpi_{n,j} Q_j[q_2, \dots, q_k, q_1; t], \\ \vdots \\ Q_n[q_k, q_1, \dots, q_{k-1}; t] = 1 + 2 \sum_{1 \leq j < n} \rho_{n,j}(q_k) \varpi_{n,j} Q_j[q_1, \dots, q_k; t] \end{cases}$$

for $n \geq 2t + 1$, with the initial values $Q_{n,t} \equiv Q_n$ for $n \leq 2t$.

Differential equations. Define the differential operator

$$J_q[\varphi](z) = \left((\vartheta + t) \cdots (\vartheta + 2t) + \frac{q(t+1)}{z} (\vartheta + t + 1) \cdots (\vartheta + 2t) \right) \varphi(z) \quad (q \in \{0, 1\}).$$

From (6.2), we derive a system of differential equations satisfied by the generating functions

$$f_j^{[t]}(z) := Q[q_j, \dots, q_k, q_1, \dots, q_{j-1}; t](z) := \sum_{n \geq 1} Q_{n,t} z^n.$$

LEMMA 6.2. Let $\lambda_t := (2t + 2)/(t + 1)!$. For any query pattern $[q_1, \dots, q_k]$,

$$\begin{cases} J_{q_1}[f_1^{[t]}] = J_{q_1}\left[\frac{z}{1-z}\right] + \lambda_t f_2^{[t]}, \\ \vdots \\ J_{q_k}[f_k^{[t]}] = J_{q_k}\left[\frac{z}{1-z}\right] + \lambda_t f_1^{[t]}. \end{cases}$$

Proof (sketch). When $q_j = 0$, we have

$$\vartheta^{\overline{2t+1}}\left(f_j^{[t]} - \frac{z}{1-z}\right) = \lambda_t \vartheta^{\bar{t}} f_{j+1}^{[t]} \quad (1 \leq j \leq k),$$

where $f_{k+1}^{[t]} := f_1^{[t]}$.

When $q_j = 1$, the differential equation becomes

$$\vartheta^{\overline{2t+2}}\left(z f_j^{[t]} - \frac{z^2}{1-z}\right) = \lambda_t \vartheta^{\overline{t+1}}(z f_{j+1}^{[t]}).$$

By applying the identity $\vartheta((1-z)\varphi(z)) = (1-z)(\vartheta-1)\varphi(z)$, we obtain

$$\left((\vartheta+t)\cdots(\vartheta+2t) + \frac{(t+1)}{z}(\vartheta+t+1)\cdots(\vartheta+2t)\right) f_j^{[t]} = \lambda_t f_{j+1}^{[t]}. \quad \square$$

COROLLARY 6.3.

$$(J_{q_k} \circ \dots \circ J_{q_1})[f_1^{[t]}] - \lambda_t^k f_1^{[t]} = \sum_{1 \leq j \leq k} \lambda_t^{j-1} (J_{q_k} \circ \dots \circ J_{q_j})\left[\frac{z}{1-z}\right].$$

Differential equations: Normal form. We multiply, as in the case of k -d trees, both sides by $z^{\mu_{\mathbf{q}}(t)}$, where $\mu_{\mathbf{q}}(t)$ denotes the “minimum exponent”

$$\mu_{\mathbf{q}}(t) := (k - v)(t + 1) + 1,$$

with v being the last position of the first block of 1’s (from left to right), and $\mu_{\mathbf{q}}(t) := 0$ if $s = 0$.

Then the generating function

$$f(z) := f_1^{[t]}(z) - \sum_{1 \leq j < k(t+1)} Q_j z^j$$

satisfies the differential equation

$$P_0(\vartheta)f = \sum_{1 \leq \ell \leq \mu_{\mathbf{q}}(t)} (1-z)^\ell P_\ell(\vartheta)f + g,$$

where the P_ℓ ’s are polynomials of degree $k(t + 1)$ and

$$\begin{aligned} g(z) = & z^{\mu_{\mathbf{q}}(t)} \sum_{1 \leq j \leq k} \lambda_t^{j-1} (J_k \circ \dots \circ J_j)\left[\frac{z}{1-z}\right](z) + z^{\mu_{\mathbf{q}}(t)} \lambda_t^k \sum_{1 \leq j < k(t+1)} Q_j z^j \\ & - z^{\mu_{\mathbf{q}}(t)} (J_k \circ \dots \circ J_1)\left[\sum_{1 \leq j < k(t+1)} Q_j z^j\right](z). \end{aligned}$$

In particular, $P_0(x)$ is as given in (6.1).

A series form for K_t . Following the same method of proof as for k -d trees, we can derive a series representation for K_t .

THEOREM 6.4. *The expected cost $Q_{n,t}$ satisfies the approximation*

$$Q_{n,t} \sim \frac{K_t}{\Gamma(\alpha)} n^{\alpha-1},$$

where

$$K_t = \frac{1}{P'_0(\alpha)} \sum_{j \geq 0} g^*(\alpha + j) B_j,$$

with the series being absolutely convergent. Here $g^*(w) := \int_0^1 g(x)(1-x)^{w-1} dx$ and B_j is defined recursively by

$$B_j = \sum_{1 \leq \ell \leq \mu_{\mathbf{q}}(t)} \frac{P_\ell(\alpha + j)}{P_0(\alpha + j)} B_{j-\ell} \quad (j \geq 1),$$

with the initial values $B_0 = 1$ and $B_j = 0$ for $j < 0$.

The method of proof is the same as that for k -d trees. In particular, the estimates $g^*(w) = O(|w|^{-\mu_{\mathbf{q}}(t)-1})$ and $B_j = O(|j|^{\mu_{\mathbf{q}}(t)-q_k-1})$ hold. Note that the generating function $B(z) := \sum_j B_j z^j$ satisfies the differential equation

$$\left(\prod_{t < \ell \leq 2t} \left(\ell + \alpha - \frac{\mu_{\mathbf{q}}(t)z}{1-z} + z\mathbb{D}_z \right) \right) \times \left(\frac{q_1(t+1)}{1-z} + t + \alpha - \frac{\mu_{\mathbf{q}}(t)z}{1-z} + z\mathbb{D}_z \right) \times \dots$$

$$\times \left(\prod_{t < \ell \leq 2t} \left(\ell + \alpha - \frac{\mu_{\mathbf{q}}(t)z}{1-z} + z\mathbb{D}_z \right) \right) \left(\frac{q_k(t+1)}{1-z} + t + \alpha - \frac{\mu_{\mathbf{q}}(t)z}{1-z} + z\mathbb{D}_z \right) B(z) = 0,$$

and the indicial equation is given by

$$\prod_{1 \leq j \leq k} \left((-\zeta - \mu + (q_{k-j+1} + j - 1)(t + 1)) \prod_{1 \leq \ell \leq t} (-\zeta - \mu + (j - 1)(t + 1) + \ell) \right),$$

with the dominant (simple) zero being $\zeta = -\mu + q_k$.

All coordinates specified: $s = k$. In this case, the two Q_n 's on both sides of (6.2) are the same; thus the k recurrences reduce to a single one. The associated differential equation is

$$\left(\vartheta^{2t+2} - \lambda_t \vartheta^{t+1} \right) (zf(z)) = \frac{(2t+2)!}{1-z},$$

with suitable initial conditions. Note that the polynomial $(x+t+1) \cdots (x+2t+1) - \lambda_t$ has 1 as the simple, largest zero (in real part). By applying [10, Theorem 1], we have

$$Q_n[1, \dots, 1; t] \sim \frac{\log n}{H_{2t+2} - H_{t+1}},$$

another well-known result; see [11, 30, 36].

Hypergeometric cases: $\mu_{\mathbf{q}}(t) = 1$. For query patterns of the form (2.15), we have $\mu_{\mathbf{q}}(t) = 1$, and

$$P_1(x) = \prod_{t \leq j \leq 2t} (x+j)^k - \lambda_t^k.$$

7. Conclusions. We derived an effective expression for the leading constant of the expected cost used for random partial match queries in random k -d trees. The proposed approach for solving linear differential equations with polynomial coefficients is very general, and its success relies on the close interplay between complex-analytic methods and elementary methods (without complex analysis).

Complex-analytic methods, when they apply, usually give efficient approximations and neat expressions, but at the cost that stronger analytic properties are needed. On the other hand, elementary methods are computationally less efficient but can provide more general results under weaker conditions. A combination of both (heuristically or rigorously) usually yields very powerful tools, as already shown by the problems studied in [7, 8, 9, 10] and in this paper.

REFERENCES

- [1] M. ABRAMOWITZ AND I. A. STEGUN, *Handbook of Mathematical Functions, with Formulas, Graphs, and Mathematical Tables*, Dover, New York, 1972.
- [2] J. L. BENTLEY, *Multidimensional binary search trees used for associative searching*, Comm. ACM, 18 (1975), pp. 509–517.
- [3] J. L. BENTLEY, *K-d trees for semidynamic point sets*, in Proceedings of the Sixth Annual Symposium on Computational Geometry, ACM, New York, 1990, pp. 187–197.
- [4] J. L. BENTLEY AND J. H. FRIEDMAN, *Data structures for range searching*, ACM Comput. Surveys, 11 (1979), pp. 397–409.
- [5] E. BERTINO, B. CATANIA, B. C. OOI, R. SACKS-DAVIS, K. L. TAN, J. ZOBEL, B. SHIDLOVSKY, AND B. CATANIA, *Indexing Techniques for Advanced Database Systems*, Kluwer Academic Publishers, Boston, 1997.
- [6] P. CHANZY, L. DEVROYE, AND C. ZAMORA-CURA, *Analysis of range search for random k-d trees*, Acta Inform., 37 (2001), pp. 355–383.
- [7] H.-H. CHERN, M. FUCHS, AND H.-K. HWANG, *Phase changes in random point quadtrees*, ACM Trans. Algorithms, to appear.
- [8] H.-H. CHERN AND H.-K. HWANG, *Phase changes in random m-ary search trees and generalized quicksort*, Random Structures Algorithms, 19 (2001), pp. 316–358.
- [9] H.-H. CHERN AND H.-K. HWANG, *Partial match queries in random quadtrees*, SIAM J. Comput., 32 (2003), pp. 904–915.
- [10] H.-H. CHERN, H.-K. HWANG, AND T.-H. TSAI, *An asymptotic theory for Cauchy-Euler differential equations with applications to the analysis of algorithms*, J. Algorithms, 44 (2002), pp. 177–225.
- [11] W. CUNTO, G. LAU, AND P. FLAJOLET, *Analysis of kdt-trees: kd-trees improved by local re-organisations*, in Algorithms and Data Structures, Lecture Notes in Comput. Sci. 382, Springer, Berlin, 1989, pp. 24–38.
- [12] L. DEVROYE, J. JABBOUR, AND C. ZAMORA-CURA, *Squarish k-d trees*, SIAM J. Comput., 30 (2000), pp. 1678–1700.
- [13] A. DUCH AND C. MARTÍNEZ, *On the average performance of orthogonal range search in multi-dimensional data structures*, J. Algorithms, 44 (2002), pp. 226–245.
- [14] P. FLAJOLET, G. GONNET, C. PUECH, AND J. M. ROBSON, *Analytic variations on quadtrees*, Algorithmica, 10 (1993), pp. 473–500.
- [15] P. FLAJOLET, X. GOURDON, AND P. DUMAS, *Mellin transforms and asymptotics: Harmonic sums*, Theoret. Comput. Sci., 144 (1995), pp. 3–58.
- [16] P. FLAJOLET AND A. M. ODLYZKO, *Singularity analysis of generating functions*, SIAM J. Discrete Math., 3 (1990), pp. 216–240.
- [17] P. FLAJOLET AND P. PUECH, *Partial match retrieval of multi-dimensional data*, J. ACM, 32 (1986), pp. 371–407.
- [18] P. FLAJOLET AND R. SEDGEWICK, *Analytic Combinatorics*, in preparation. Preliminary version available online at <http://algo.inria.fr/flajolet/Publications/publist.html>.
- [19] J. H. FRIEDMAN, J. L. BENTLEY, AND R. A. FINKEL, *An algorithm for finding best matches in logarithmic expected time*, ACM Trans. Math. Software, 3 (1977), pp. 209–226.
- [20] G. H. GONNET AND R. BAEZA-YATES, *Handbook of Algorithms and Data Structures*, Addison-Wesley, Reading, MA, 1991.
- [21] A. G. GRAY AND A. W. MOORE, *“N-body” problems in statistical learning*, in Advances in Neural Information Processing Systems, T. K. Leen and T. G. Dietterich, eds., MIT Press, Cambridge, MA, 2001, pp. 521–527.

- [22] P. J. GROTHER, G. T. CANDELA, AND J. L. BLUE, *Fast implementations of nearest neighbor classifiers*, Pattern Recognition, 30 (1997), pp. 459–465.
- [23] T. N. HIBBARD, *Some combinatorial properties of certain trees with applications to searching and sorting*, J. ACM, 9 (1962), pp. 13–28.
- [24] E. L. INCE, *Ordinary Differential Equations*, Dover, New York, 1926.
- [25] D. T. LEE AND C. K. WONG, *Worst-case analysis for region and partial region searches in multidimensional binary search trees and quad trees*, Acta Inform., 9 (1977), pp. 23–29.
- [26] C. MARTÍNEZ, A. PANHOLZER, AND H. PRODINGER, *Partial match queries in relaxed multidimensional search trees*, Algorithmica, 29 (2001), pp. 181–204.
- [27] R. NEININGER, *Asymptotic distributions for partial match queries in K - d trees*, Random Structures Algorithms, 17 (2000), pp. 403–427.
- [28] R. C. NICHOL, S. CHONG, A. J. CONNOLLY, S. DAVIES, C. GENOVESE, A. M. HOPKINS, C. J. MILLER, A. W. MOORE, D. PELLEG, G. T. RICHARDS, J. SCHNEIDER, I. SZAPUDI, AND L. WASSERMAN, *Computational AstroStatistics: Fast and Efficient Tools for Analysing Huge Astronomical Data Sources*, preprint, 2002. Available online at <http://arxiv.org/abs/astro-ph/0110230>.
- [29] S. M. OMOHUNDRO, *Efficient algorithms with neural network behavior*, Complex Systems, 1 (1987), pp. 273–347.
- [30] P. V. POBLETE AND J. I. MUNRO, *The analysis of a fringe heuristic for binary search trees*, J. Algorithms, 6 (1985), pp. 336–350.
- [31] F. P. PREPARATA AND M. I. SHAMOS, *Computational Geometry: An Introduction*, Springer-Verlag, New York, 1985.
- [32] J. REISS, J.-J. AUCOUTURIER, AND M. SANDLER, *Efficient multidimensional searching routines for music information retrieval: Use k - d trees to retrieve music information*, in Proceedings of the Second Annual International Symposium on Music Information Retrieval (Bloomington, IN), J. S. Downie and D. Bainbridge, eds., 2001, pp. 163–171.
- [33] H. SAMET, *The Design and Analysis of Spatial Data Structures*, Addison-Wesley, Reading, MA, 1990.
- [34] H. SAMET, *Applications of Spatial Data Structures: Computer Graphics, Image Processing, and GIS*, Addison-Wesley, Reading, MA, 1990.
- [35] F. SCHWARZER AND I. LOTAN, *Approximation of protein structure for fast similarity measures*, in Proceedings of the Seventh Annual ACM International Conference on Computational Biology (RECOMB), ACM, New York, 2003, pp. 267–276.
- [36] R. SEDGEWICK, *Quicksort*, Garland Publishing, New York, 1980.
- [37] S. S. SKIENA, *Who is interested in algorithms and why? Lessons from the Stony Brook algorithms*, ACM SIGACT News, 30 (1999), pp. 65–74. Available online at <http://www.cs.sunysb.edu/~skiena/hits>.

POWER FROM RANDOM STRINGS*

ERIC ALLENDER[†], HARRY BUHRMAN[‡], MICHAL KOUCKÝ[§], DIETER VAN
MELKEBEEK[¶], AND DETLEF RONNEBURGER^{||}

Abstract. We show that sets consisting of strings of high Kolmogorov complexity provide examples of sets that are complete for several complexity classes under probabilistic and nonuniform reductions. These sets are provably not complete under the usual many-one reductions.

Let $R_C, R_{K_t}, R_{KS}, R_{KT}$ be the sets of strings x having complexity at least $|x|/2$, according to the usual Kolmogorov complexity measure C , Levin’s time-bounded Kolmogorov complexity K_t [L. Levin, *Inform. and Control*, 61 (1984), pp. 15–37], a space-bounded Kolmogorov measure KS , and a new time-bounded Kolmogorov complexity measure KT , respectively.

Our main results are as follows:

1. R_{KS} and R_{K_t} are complete for PSPACE and EXP, respectively, under P/poly-truth-table reductions. Similar results hold for other classes with PSPACE-robust Turing complete sets.
2. $EXP = NP^{R_{K_t}}$.
3. $PSPACE = ZPP^{R_{KS}} \subseteq P^{R_C}$.
4. The Discrete Log, Factoring, and several lattice problems are solvable in $BPP^{R_{KT}}$.

Our hardness result for PSPACE gives rise to fairly natural problems that are complete for PSPACE under \leq_T^P reductions, but not under \leq_m^{\log} reductions.

Our techniques also allow us to show that all computably enumerable sets are reducible to R_C via P/poly-truth-table reductions. This provides the first “efficient” reduction of the halting problem to R_C .

Key words. Kolmogorov complexity, completeness, reductions, randomness, derandomization

AMS subject classifications. 68Q30, 68Q15, 68Q17, 68Q10, 03D15

DOI. 10.1137/050628994

1. Introduction. Much recent work in derandomization can be viewed as an attempt to understand and exploit the interplay between the two common meanings of the phrase “random string”: a string picked at random (according to some distribution), and a string with high Kolmogorov complexity (in some sense). In this paper, we further investigate the relationship between these two notions. We apply recent advances in derandomization to obtain fundamentally new types of complete sets for several standard complexity classes. The sets consist of random strings with respect to various Kolmogorov measures.

*Received by the editors April 11, 2005; accepted for publication (in revised form) October 17, 2005; published electronically April 21, 2006. This paper combines work presented at the 24th *Conference on Foundations of Software Technology and Theoretical Computer Science* [6] and at the 43rd *IEEE Annual Symposium on Foundations of Computer Science* [9].

<http://www.siam.org/journals/sicomp/35-6/62899.html>

[†]Rutgers University, New Brunswick, NJ 08855 (allender@cs.rutgers.edu). The work of this author was partially supported by NSF grant CCR-0104823.

[‡]CWI and University of Amsterdam, Amsterdam, 1090 GB, The Netherlands (buhrman@cwi.nl).

[§]McGill University, Montréal, PQ, H3A 2T5, Canada (mkoucky@cs.mcgill.ca). The work of this author was partially supported by NSF grant CCR-0104823. Part of this work was done while the author was a graduate student at Rutgers University, NJ.

[¶]University of Wisconsin, Madison, WI 53706 (dieter@cs.wisc.edu). The work of this author was partially supported by NSF Career Award CCR-0133693.

^{||}York College, CUNY, Jamaica, NY 11451 (ronnebg@york.cuny.edu). The work of this author was partially supported by NSF grant CCR-0104823 and PSC-CUNY Award 60112-35 36. Part of this work was done while the author was a graduate student at Rutgers University, NJ.

We will focus on the set $R_C = \{x : C(x) \geq |x|/2\}$ of strings with high traditional Kolmogorov complexity, as well as various resource-bounded variants R_μ for $\mu = \text{KT}, \text{KS}, \text{Kt}$. See section 2 for the definitions of KT, KS , and Kt . The choice of $|x|/2$ as a quantification of “high complexity” is rather arbitrary. Our results hold for any reasonable bound ranging from $|x|$ to $|x|^\epsilon$ for any positive ϵ . In most cases, our results carry over to other notions of resource-bounded Kolmogorov complexity that have been considered in the literature.

The sets R_μ of Kolmogorov random strings are good examples of sets with a lot of information content that is difficult to access. There are many examples in the literature where sets consisting of strings with high resource-bounded Kolmogorov complexity have been studied as possible examples of intractable sets that are not complete for any of the standard complexity classes. We list a few of these results here.

1. Buhrman and Mayordomo [20] studied a time-bounded Kolmogorov complexity measure K^t for exponential time bounds $t(n) \geq 2^{n^2}$, and they showed that the set $R^t = \{x : K^t(x) \geq |x|\}$ lies in $\text{EXP} - \text{P}$ and is *not* complete for EXP under polynomial-time Turing reducibility $\leq_{\text{T}}^{\text{P}}$.

2. Ko [38] studied the measure K^t for *polynomial* time bounds $t(n)$ and showed that the questions of whether the set $\{x : K^t(x) \geq \epsilon|x|\}$ is in P or is coNP -complete (with respect to $\leq_{\text{T}}^{\text{P}}$ reductions) cannot be answered using relativizing techniques.

3. Kabanets and Cai [36] study the Minimum Circuit Size Problem (MCSP) defined as the set of all pairs (χ_f, s) where χ_f denotes a string of length 2^n that is the truth-table of a Boolean function f on n variables and s denotes an integer such that f can be computed by a Boolean circuit of size at most s . Because the circuit size of f is polynomially related to $\text{KT}(f)$ (see section 2), the MCSP can be seen to be related to R_{KT} . Kabanets and Cai present evidence that the MCSP is not in P but is also not likely to be NP -complete under $\leq_{\text{m}}^{\text{P}}$ reductions.

4. In the traditional setting of Kolmogorov complexity, where no resource bounds are present, the set of Kolmogorov random strings R_C is easily seen to be co-computably enumerable (co-c.e.) and not decidable, but the complement of the halting problem is *not* reducible to R_C via a many-one reduction. It was shown only within the last decade by Kummer that a truth-table reduction is sufficient to reduce the complement of the halting problem to R_C [40], although it had long been known [46] that Turing reductions can be used. It should be emphasized that Kummer’s reduction, which is in fact a disjunctive truth-table reduction, is *not* feasible and asks *many* queries. It can be shown that R_C is not complete for the co-c.e. sets under polynomial-time disjunctive truth-table reductions. (This was originally observed in [9]; a more general statement is proved in [8].)

These results suggest that the sets of resource-bounded random strings are not complete for the complexity class they naturally live in. Buhrman and Torenvliet [21] gave some evidence that this is not the complete picture. They showed that for the *conditional* version of space-bounded Kolmogorov complexity, the set of random strings is hard for PSPACE under NP reductions. However, their result had the major drawback that it needed conditional Kolmogorov complexity and used NP reductions, and, moreover, their proof technique could not be used beyond PSPACE .

Using very different techniques, we provide much stronger results in the same direction. We show that the set of random strings *can* be exploited by efficient reductions. For instance, we show that the set R_{Kt} of strings with high complexity using Levin’s time-bounded Kolmogorov notion Kt [43] is complete for EXP under truth-table reductions computable by polynomial-size circuits. Since this set is provably

not complete under polynomial-time many-one reductions, we obtain natural examples that witness the difference in power of various reducibilities.

It is significant that all of our completeness results are obtained via derandomization techniques. This provides a new paradigm for proving completeness in settings where more traditional methods provably are not applicable.

In some instances, we are also able to provide completeness results under uniform reductions. By making use of multiple-prover interactive proofs for EXP [13] we show that R_{Kt} is complete for EXP under NP-Turing reductions.

Of greater interest is the fact that the set R_{KS} of strings with high space-bounded Kolmogorov complexity is complete for PSPACE under ZPP-Turing reductions. Our proofs rely on the existence of complete sets for PSPACE that are both downward self-reducible and random self-reducible [57], and hence our proofs do not relativize. It remains unknown whether the results themselves hold relative to all oracles.

For the unbounded case we even show that PSPACE is reducible to R_{C} under *deterministic* polynomial-time Turing reductions. The main tool in proving this is a polynomial-time algorithm that constructs a string of high Kolmogorov complexity, using R_{C} as an oracle.

For R_{KT} , a set in coNP, we do not obtain completeness results but we show that R_{KT} is hard under BPP-reductions for some well-known candidate NP-intermediate problems: Discrete Log, Factoring, and several lattice problems. These hardness results also hold for the minimum circuit size problem (MCSP), thereby improving results of [36].

1.1. The connection to derandomization. There is an obvious connection between Kolmogorov complexity and derandomization. For any randomized algorithm \mathcal{A} having small error probability, and any input x , the coin flip sequences r resulting in a wrong answer are atypical and hence have short descriptions of some kind.

By considering different notions of Kolmogorov complexity, less obvious (and more useful) connections can be exposed. Assume that the coin flip sequences that result in wrong answers have small μ -complexity, for some Kolmogorov measure μ . If we can generate a coin flip sequence r belonging to the set R_{μ} of strings with high μ -complexity, then we obtain an upper bound on the complexity of \mathcal{A} , by running the randomized algorithm on (x, r) . Instead of generating a string in R_{μ} , if we assume only that we can check membership in R_{μ} , then we obtain a zero-error probabilistic algorithm by picking a coin flip sequence r at random until we get one in R_{μ} (of which there are many) and then running \mathcal{A} on (x, r) .

The first interesting application of this approach is due to Sipser [54]. His proof that BPP lies in the polynomial-time hierarchy uses $\mu = \text{KD}^t$, the $t(n)$ -time-bounded distinguishing complexity, for some polynomial t . The corresponding set R_{μ} lies in Π_2^p . The hardness versus randomness tradeoffs by Babai et al. [14] and by Impagliazzo and Wigderson [33] and the “easy witness technique” of [35, 32] can be cast as an application of this approach with $\mu = \text{KT}$. A survey of some derandomization results from this perspective can be found in [6]. The observation from [37] that the construction of [33] relativizes with respect to any oracle A can be viewed in terms of a Kolmogorov measure which we denote as KT^A . This interpretation plays a crucial role in Trevisan’s influential construction of extractors out of pseudorandom generators [56].

Our main technique is to use relativizing hardness versus randomness tradeoffs in the contrapositive. Such results state that if there exists a computational problem in a certain complexity class \mathcal{C} that is hard when given oracle access to A , then there exists

a pseudorandom generator secure against A that is computable within \mathcal{C} . However, we argue that no pseudorandom generator computable in \mathcal{C} can be secure against R_μ . Thus we conclude that every problem in \mathcal{C} is easy given oracle access to R_μ ; i.e., \mathcal{C} reduces to R_μ . For our results, we exploit the nonuniform hardness versus randomness tradeoffs in [14] and [33], as well as the uniform ones in [30] and [34].

1.2. The structure of complexity classes. The tools of reducibility and completeness are responsible for most of the success that complexity theory has had in proving (or providing evidence for) intractability of various problems. It has been known since the work of Ladner [41] that, if P is not equal to NP, then there are intractable problems in NP that are not NP-complete. There are not many interesting candidates for this status, though. Certainly the sets constructed in [41] are quite artificial; they are constructed by putting huge empty segments inside a standard complete problem such as SAT. Similarly, there are a great many notions of reducibility that have been considered and for many of these notions it is known that more powerful reducibilities provide more complete sets (at least for large complexity classes such as EXP) [61, 2]. However, almost all of the known constructions proceed by diagonalization and do not produce very “natural” languages.

We have already mentioned three notable examples that run counter to this trend [20, 36, 38]. Using the insight that circuit size is closely connected to a version of time-bounded Kolmogorov complexity, all of these examples can be seen to be variations on a single theme. Our results amplify and extend these earlier papers. For instance, the set R^t (for $t = 2^{n^2}$) was shown in [20] not to be complete for EXP under \leq_T^P reductions, but the authors presented no positive results showing how to reduce problems in EXP to R^t . In contrast, we show that this set is complete under $\leq_{tt}^{P/poly}$ and \leq_T^{NP} reductions. To the best of our knowledge, this is the first example of a “natural” set A with the property that $P^A \neq NP^A$. All previous examples of such sets have been explicitly constructed via diagonalization, to separate P and NP. For polynomial time bounds t , Ko raised the prospect that R^t might be complete for NP under \leq_T^{SNP} reductions, but he presented no unconditional results reducing supposedly intractable problems to R^t . In contrast, we show that factoring and various problems arising in cryptography are reducible to R^t (and in many cases we present \leq_T^{ZPP} reductions).

As an additional application of our techniques toward explicating the structure of complexity classes, we present natural examples of sets that witness the difference of \leq_{tt}^P and \leq_T^{\log} or \leq_T^P and \leq_m^P completeness notions in PSPACE.

1.3. Outline of the rest of the paper. In section 2 we present background and definitions regarding resource-bounded Kolmogorov complexity. In section 3 we present our results for R_{Kt} , R_{KS} , and R_C , and in section 4 those for R_{KT} . We conclude with open problems in section 5.

2. Resource-bounded Kolmogorov complexity. In this section, we present the notions of resource-bounded Kolmogorov complexity that we use in stating most of our results. Before presenting our definitions, it is necessary to give some justification for introducing new definitions, because it might seem to the reader that the literature has more than enough notions of resource-bounded Kolmogorov complexity already.

2.1. Motivation. Kolmogorov complexity provides a very useful and elegant tool for measuring the complexity of finite objects. We refer the reader to [44] for the standard theorems and notation related to Kolmogorov complexity. Since the Kolmogorov complexity $C(x)$ of a string x cannot be computed, many definitions have been proposed for computable approximations to $C(x)$.

One of the most useful of these was given by Levin [43]. Given a universal Turing machine U , Levin defined $\text{Kt}(x|y)$ to be $\min\{|d| + \log t : U(d, y) = x \text{ in at most } t \text{ steps}\}$ and $\text{Kt}(x)$ to be a shortcut for $\text{Kt}(x|\lambda)$, where λ denotes the empty string. Levin's Kt measure has many useful properties, among which is the fact that it provides a search strategy for finding accepting computations for nondeterministic Turing machines that provably is nearly optimal. Stated another way, suppose that you want to find satisfying assignments for Boolean formulae. Given a formula ϕ with n variables, you want to find a string $v \in \{0, 1\}^n$ that satisfies ϕ . If this problem can be solved in time $t(n)$, then it can be solved in time nearly $O(t(n))$ by a program that, on input ϕ , enumerates strings v in order of increasing value of $\text{Kt}(v|\phi)$. For details, see [44].

For reasons of ease of presentation, we will slightly change the formal definition of Kt in section 2.2. The modification affects the value of Kt by at most a logarithmic additive term and does not alter any of Kt 's desirable properties. The principal change is in the way the universal machine U uses the description d . In Levin's original definition, U uses the description d to output the entire string x . In the new definition, the description d allows U to compute each bit of x . That is, given d and i , U can compute the i th bit of x . Although this causes no substantial change to the measure Kt , it opens up the possibility of considering run times that are much smaller than the length of the string x . In turn, this allows us to introduce a new notion of time-bounded Kolmogorov complexity, denoted KT , which is very closely related to circuit complexity. More precisely, $\text{KT}(x)$ is polynomially related to the minimum circuit size of the function that has x as its truth-table (see Theorem 11).

Other notions of resource-bounded Kolmogorov complexity have been studied previously. One definition that one encounters fairly frequently [38, 45, 44, 20, 21] is the measure $\text{K}^t(x)$, in which one explicitly bounds the running time of the universal machine U by $t(|x|)$, where the function t is an additional parameter. In contrast to our notion KT , there is no known relationship between circuit size and K^t . The need to fix the running time t also makes the notion K^t less robust with respect to the choice of the underlying universal machine U . See section 2.2 for more details.

It turns out to be convenient to consider relativized measures KT^A where the universal machine is given access to an oracle A . This highlights some close connections with some previously studied Kolmogorov complexity measures. For instance,

1. Levin's measure $\text{Kt}(x)$ is linearly related to $\text{KT}^A(x)$, where A is a certain complete set for E ; and
2. the original Kolmogorov measure $\text{C}(x)$ is linearly related to $\text{KT}^H(x)$, where H is the halting problem.

The relationship between KT and circuit size also holds in the relativized setting. Thus Levin's measure Kt is roughly the same thing as circuit size on oracle circuits that have access to an E -complete set, and C is roughly the same thing as circuit size relative to the halting problem.

The computational model that we use throughout this paper is the multitape Turing machine with random access to its input tape. Our ideas work in any general model of computation. The main reason for providing random access to the input tape is that $\text{KT}(x|y)$ can be seen to be closely related to the circuit size of the function represented by x on circuits that have oracle access to the function represented by y . For this correspondence to hold, U must be able to access any bit of y quickly. Let us emphasize that all of our theorems in sections 3 and 4 hold verbatim for models such as Turing machines with sequential access to their input tapes or Turing machines that have random access to all their tapes.

2.2. Formal definitions. We begin by defining the models of computation used in this paper and by observing some basic properties relating them to one another.

DEFINITION 1. We consider circuits with AND, OR, and negation gates, as well as oracle gates for one or more oracles. An oracle gate for an oracle A has inputs z_1, \dots, z_r for some r ; it outputs the value 1 if the string $z = z_1, \dots, z_r$ is in A , and outputs 0 otherwise. We define the size of a circuit as the number of connections in the circuit.

We also associate circuit complexities with Boolean strings viewed as truth-tables of Boolean functions.

DEFINITION 2. Let x be a string in $\{0, 1\}^*$ of length n . Let $2^{k-1} < n \leq 2^k$ and let x' be the string of length 2^k of the form $x0^i$ for some i . Let f_x be the Boolean function on k variables, having x' as its truth-table. We define $\text{SIZE}(x)$ to be the size of a minimal-sized circuit with k input variables computing the function f_x . For any oracle A , $\text{SIZE}^A(x)$ denotes the circuit size required to compute f_x on circuits with oracle A . If $y \in \{0, 1\}^*$, then $\text{SIZE}^{A,y}(x)$ denotes the circuit size required to compute f_x on circuits with two kinds of oracle gates: oracle gates for A of arbitrary fan-in and oracle gates for f_y of fan-in $\lceil \log |y| \rceil$.

For a good overview of circuit complexity refer to [60].

We have already mentioned that we will be using a Turing machine with random access to its input tape; we now make more precise what we mean by this. We use essentially the same notion that was considered in [15]. The machine has one read-only input tape of length n , a constant number of read-write working tapes of infinite length, and a read-write input address tape. At every time step the machine can modify the content of its read-write tapes using the appropriate heads and move these heads left or right by one tape cell. It can also query the content of the input bit whose address is written on the input address tape. If there is no such input bit the reply to the query is the symbol “*.”

In the case where the machine is an oracle Turing machine, for each oracle the machine has one read-write oracle tape. At every step the machine can query any of its oracles whether the string written on the corresponding oracle tape belongs to the oracle set or not. We also allow finite oracles. For a finite oracle $y \in \{0, 1\}^*$, the machine obtains as an answer to its query i bit y_i if $i \leq |y|$ and “*” otherwise. Note that the input tape behaves like an oracle tape accessing a finite oracle.

As usual, the *running time* of a Turing machine on a particular input is the number of steps before a computation of the machine stops on that input, and the *space* used during the computation is the number of read-write tape cells visited during the computation.

As with ordinary Turing machines, running time on our machines is closely related to circuit size.

PROPOSITION 3 (simulation of circuits). Let $k \geq 0$ be an integer. There is a constant c and a two-tape oracle Turing machine M such that for every oracle A , finite oracles y_1, \dots, y_k , and circuit C^{A,y_1,\dots,y_k} of size m computing a function of n input bits, there is an encoding d_C of C of size $\leq cm(\log m + \log n)$, so that for any string x , if $|x| = n$ then $M^{A,d_C,y_1,\dots,y_k}(x)$ outputs $C^{A,y_1,\dots,y_k}(x)$ in time $\leq c(m^2 \log m + m \log n)$, and it outputs * in time $\leq c(\log n)$ otherwise.

PROPOSITION 4 (simulation by circuits). For every oracle Turing machine M there is a constant c_M such that if M^{A,y_1,\dots,y_k} runs in time $t = t(n, n_1, \dots, n_k)$ on inputs of length n with oracles y_1, \dots, y_k of length n_1, \dots, n_k , respectively, then for every n and n_1, \dots, n_k there is a circuit C_{n,n_1,\dots,n_k} of size $c_M t(t^2 + n + \log \sum_{i=1}^k n_i)$, such that $C^{A,y_1,\dots,y_k}(x) = M^{A,y_1,\dots,y_k}(x)$ for all inputs x of length n .

Using the technique of Hennie and Stearns [31] and Fürer [26, 27] we can establish the following proposition.

PROPOSITION 5 (minimal simulation overhead). *There is a Turing machine U with two work tapes, such that for any oracle Turing machine M there is a constant c_M so that for any oracle A and finite oracles d and y and input x , there is a finite oracle d' of length at most $|d| + c_M$ such that $U^{A,d',y}(x) = M^{A,d,y}(x)$. The computation time of U is at most $c_M t \log t$ and the space used is at most $c_M s$, where $M^{A,d,y}(x)$ runs for time t and uses space s . Furthermore, if M is a two-tape machine, then the running time of U is bounded by $c_M t$.*

We call any machine U that satisfies the previous proposition a *universal Turing machine*; note that we require our universal Turing machines to be efficient in simulating other machines.

DEFINITION 6. *A Turing machine U is universal if it satisfies all properties stated in Proposition 5.*

It is customary to define the Kolmogorov complexity function $C(x)$ to be the length of the shortest description $d \in \{0, 1\}^*$ such that $U(d) = x$, where U is a universal Turing machine. In order to present all of our definitions in a uniform framework, we give a slightly different definition below. We use the notation $x_{|x|+1} = *$ to denote the “end of string marker.”

DEFINITION 7 (Kolmogorov complexity). *Let U be a Turing machine and let A be an oracle. Define*

$$C_U^A(x|y) = \min\{|d| : \forall b \in \{0, 1, *\} \forall i \leq |x| + 1, \text{ the machine } U^{A,d,y}(i, b) \text{ accepts iff } x_i = b\}.$$

We let $C_U(x|y)$ denote $C_U^\emptyset(x|y)$, and we let $C_U(x)$ denote $C_U(x|\lambda)$.

Definition 7 deviates from the standard definition in two respects. First, the universal machine U gets the index i of a bit position of the string x as input and needs only to determine x_i , the i th bit of x . Since this mechanism alone does not encode the length of x , we stipulate that we obtain $*$ for bit position $i = |x| + 1$. As discussed in section 2.1, this change allows us to consider sublinear running times for U , which will be critical for the new Kolmogorov measure KT we introduce.

Second, the way we define the value of x_i is through a distinguishing process; i.e., the machine U needs only to recognize the correct value of x_i . For the results in our paper, we could as well use the traditional defining mechanism, which calls for the machine U to produce x_i . In fact, we used that convention in our original definition of the measure KT [6, 9]. However, it has subsequently been useful to consider new measures in this framework, where nondeterministic and alternating machines U are also taken into account [11, 7]. In order that all of these measures can be presented in a uniform framework, we adopt the convention that the machine U needs only to recognize the correct value of x_i . That is, given description d , index i , and $b \in \{0, 1, *\}$, U^d should accept (i, b) if and only if b equals x_i .

The reader can easily check that Definition 7 is exactly equivalent to the usual one. That is, if d is a description of x in the “traditional” definition, then d will also be a description of x in Definition 7, for a suitable machine U .

If U is a universal machine, we have the useful property that for any other machine U' there is a constant c such that $C_U^A(x|y) \leq C_{U'}^A(x|y) + c$ for all x, y , and A . Following convention (as in [44]), we pick one such universal machine U and define $C^A(x|y)$ to be equal to $C_U^A(x|y)$. $C(x|y)$ and $C(x)$ are defined similarly in terms of U .

Now we present a formal definition of Levin’s time-bounded Kolmogorov complexity measure K_t using this framework.

DEFINITION 8 (Kt). Let U be a Turing machine and let A be an oracle. Define the measure $\text{Kt}_U^A(x|y)$ of a string x as

$$\text{Kt}_U^A(x|y) = \min\{|d| + \log t : \forall b \in \{0, 1, *\} \forall i \leq |x| + 1, \text{ the machine } U^{A,d,y}(i, b) \text{ accepts in } t \text{ steps iff } x_i = b\}.$$

We use $\text{Kt}_U(x|y)$ to denote $\text{Kt}_U^\emptyset(x|y)$, and we use $\text{Kt}_U(x)$ to denote $\text{Kt}_U(x|\lambda)$.

Our definition is essentially equivalent to Levin's original one up to a logarithmic term. More precisely, for suitable choices of the underlying machines, Definition 8 yields a value that is between Levin's and Levin's plus $\log|x|$.¹

As in the case of resource-unbounded Kolmogorov complexity, the measure Kt_U is essentially invariant under the particular choice of universal machine U . Now, however, instead of an additive constant term, it seems that we must accept an additive logarithmic term. Universal Turing machines can simulate any other Turing machine M in time $O(t \log t)$ (Proposition 5), where t is the original time required for the computation of M . We pick one such universal machine U and define $\text{Kt}^A(x|y)$ to be equal to $\text{Kt}_U^A(x|y)$. $\text{Kt}(x|y)$ and $\text{Kt}(x)$ are defined similarly in terms of U .

Another Kolmogorov measure that has been studied before is K^t , where the running time of the underlying machine U is bounded by a parameter t .

DEFINITION 9 (K^t). Let U be a Turing machine and t a time bound. Define the measure $\text{K}_U^t(x)$ of a string x as

$$\text{K}_U^t(x) = \min\{|d| : \forall b \in \{0, 1, *\} \forall i \leq |x| + 1, \text{ the machine } U^d(i, b) \text{ accepts in } t(|x|) \text{ steps iff } x_i = b\}.$$

In contrast to all other Kolmogorov measures that we consider, changing the choice of universal Turing machine might cause a huge change in the value of $\text{K}^t(x)$. Nonetheless, we follow the usual convention [38, 45, 44, 20, 21] and pick some universal machine U and define $\text{K}^t(x)$ to be $\text{K}_U^t(x)$. Proposition 5 guarantees that for any other choice U' of universal machine, there is a constant c such that $\text{K}^{ct \log t}(x) \leq \text{K}_{U'}^t(x)$ (although we observe that $\text{K}^{ct \log t}(x)$ might be *much* less than $\text{K}_{U'}^t(x)$). (In this paper, we will consider neither $\text{K}^t(x|y)$ nor K^t relative to oracles A .)

It is now time to give the formal definition of the measure KT that will be one of our main objects of study. This definition can be seen as a variant on a definition introduced by Antunes, Fortnow, and van Melkebeek in [12].

DEFINITION 10 (KT). Let U be a Turing machine and A an oracle. Define the measure $\text{KT}_U^A(x|y)$ to be

$$\text{KT}_U^A(x|y) = \min\{|d| + t : \forall b \in \{0, 1, *\} \forall i \leq |x| + 1, \text{ the machine } U^{A,d,y}(i, b) \text{ accepts in } t \text{ steps iff } x_i = b\}.$$

We omit the superscript A if $A = \emptyset$, and the string y if $y = \lambda$.

¹As discussed briefly in [44], Levin's original definition of Kt was formulated in terms of "Kolmogorov-Uspensky machines." This model of computation is one of several that allow for the existence of universal machines that can simulate any other algorithm with at most a linear slow-down; see also [16]. This allows for a more elegant statement of certain theorems regarding Kt complexity. For instance, using this model of computation, one can show that for a universal machine U , for every machine M $\text{Kt}_U(x) \leq \text{Kt}_M(x) + O(1)$, and searching for satisfying assignments y to a Boolean formula ϕ in order of increasing $\text{Kt}(y|\phi)$ can be shown to be optimal up to a *linear* slow-down. However, Kolmogorov-Uspensky machines confer no special advantages when considering more refined measures such as KS and KT , and thus we choose to use the more familiar Turing machines in defining our measures.

Note that the only difference between KT and Levin’s measure Kt is that in KT the time bound is given exponentially more weight than in Kt. The capital “T” in the notation for KT reflects the fact that the time component weighs more strongly than in the Kt measure.

Along with the greater emphasis on running time, we unfortunately incur a greater sensitivity on the underlying machine U . Definition 6 implies that for any universal Turing machine U and any other machine U' , $\text{KT}_U^A(x|y) \leq c \cdot \text{KT}_{U'}^A(x|y) \log \text{KT}_{U'}^A(x|y)$. We pick one such universal Turing machine U (guaranteed to exist by Proposition 5) and define $\text{KT}^A(x|y)$ to be equal to $\text{KT}_U^A(x|y)$. $\text{KT}(x|y)$ and $\text{KT}(x)$ are defined similarly in terms of U .

If one views a given string x as the truth table of a function f_x , then $\text{KT}(x|y)$ roughly corresponds to circuit complexity of f_x on circuits that have oracle access to f_y . This is made precise below.

THEOREM 11. *There is a constant c such that for any oracle A and any strings x and y in $\{0, 1\}^*$,*

- (i) $\text{Size}^{A,y}(x) \leq c(\text{KT}^A(x|y))^2 ((\text{KT}^A(x|y))^2 + \log |x| + \log |y|)$; and
- (ii) $\text{KT}^A(x|y) \leq c(\text{Size}^{A,y}(x))^2 (\log \text{Size}^{A,y}(x) + \log \log |x|)$.

Proof. To prove (i), let c_1, c_2 be suitably large constants. Assume a given string x of length n has $\text{KT}^A(x|y) = m$. Thus there is a description d_x of length at most m , such that $U^{A,d_x,y}(i, b)$ accepts if and only if $b = x_i$ in at most m steps. By Proposition 4, there is a circuit $C_{n,m}^{A,d_x,y}$ of size $c_1 m(m^2 + \log n + \log(m + |y|))$ such that $C_{n,m}^{A,d_x,y}(i, b) = U^{A,d_x,y}(i, b)$ for all i and b . The finite oracle d_x can be replaced by a circuit of size $c_2 m$ to obtain a desired circuit for x of size $cm^2(m^2 + \log n + \log(m + |y|))$, provided that $c \geq c_1 c_2$.

To prove (ii), let c_1 and M be the constant and machine given by Proposition 3. Assume that there is a circuit C of size m with oracle gates for A and y , such that on input i (in binary) the circuit computes $C(i) = x_i$ for $i \leq |x|$. Denote $n = |x|$.

By Proposition 3 there is an encoding d_C of C of size $c_1 m(\log m + \log \log n)$ such that machine M , given oracle access to A, d_C , and y , outputs $C(i)$ in time $c_1 m^2 \log m + m \log \log n$. From M one can easily construct a two-tape machine M' such that $M'^{A,d_C,y}(i, b)$ accepts if $x_i = b$ and that runs in essentially the same time as M . Thus $\text{KT}_{M'}^A(x|y) \leq c_2 m^2(\log m + \log \log n)$ for some $c_2 \geq c_1$.

The theorem now follows by Definition 6 and the fact that the defining Turing machine U for KT is universal. \square

So far we have considered time-bounded Kolmogorov complexity measures. In section 3 we will see how these measures give rise to complete problems for time-bounded complexity classes. It is useful to define space-bounded notions that will yield complete problems for space-bounded complexity classes.

DEFINITION 12 (KS). *Let U be a Turing machine and A be an oracle. We define the following space-bounded complexity measure.*

$$\text{KS}_U^A(x|y) = \min\{ |d| + s : \forall b \in \{0, 1, *\} \forall i \leq |x| + 1, \text{ the machine } U^{A,d,y}(i, b) \text{ accepts in } s \text{ space iff } x_i = b \}.$$

We omit the superscript A if $A = \emptyset$, and the string y if $y = \lambda$.

As usual, we will fix a universal Turing machine U and then drop the subscript U when considering this space-bounded Kolmogorov complexity measure. Similar to the measure KT, the measure KS is somewhat sensitive to the exact choice of the

universal machine U . However, regardless of the particular choice of universal Turing machine, Proposition 5 implies that for any Turing machine U' , there is a constant c such that $\text{KS}^A(x|y) \leq c\text{KS}_{U'}^A(x|y)$.

As is the case with traditional Kolmogorov complexity, for each of these resource-bounded Kolmogorov complexity measures KT , KS , Kt , every string has a description that is not much longer than itself.

PROPOSITION 13. *There is a constant c , such that for all strings x : $\text{KT}(x) \leq |x| + c \log |x|$, $\text{KS}(x) \leq |x| + c \log |x|$, and $\text{Kt}(x) \leq |x| + c \log \log |x|$.*

It is useful to observe that the measures C , Kt , and KS can be approximated in terms of KT^A for appropriate choices of oracle A .

THEOREM 14. *Let H denote the halting problem. There exist a complete set A for E , a complete set B for $\text{DSPACE}(n)$, and a constant c such that*

- (i) $\frac{1}{c}\text{Kt}(x) \leq \text{KT}^A(x) \leq c \cdot \text{Kt}(x)$,
- (ii) $\frac{1}{c}\text{KS}(x) \leq \text{KT}^B(x) \leq c \cdot \text{KS}(x)$, and
- (iii) $\frac{1}{c}\text{C}(x) \leq \text{KT}^H(x) \leq c \cdot (\text{C}(x) + \log |x|)$.

Proof. Let us prove first the relation between Kt and KT^A . The proof for KS is similar. Then we will consider the claim for C .

Let $A \in \text{E}$ and let x be given, such that $\text{KT}^A(x) = m$. Thus, there is a description d_x of length $|d_x| \leq m$, such that $U^{A,d_x}(i, b)$ accepts if and only if $x_i = b$ in time at most m . During the computation, U can ask queries of length at most m , and since $A \in \text{E}$, each such query can be answered in time $2^{O(m)}$. If M denotes the algorithm that simulates the computation of $U^{A,d_x}(i, b)$ for every i by directly computing the answers to the oracle queries to A , then the description $d'_x = \langle M, d_x \rangle$ is sufficient for U to compute $U^{d'_x}(i, b)$ in time $2^{O(m)}$. As $|d'_x| = m + O(1)$, we can conclude that $\text{Kt}(x) \leq m + O(1) + \log(2^{O(m)}) = O(m)$.

For the other inequality, let $A = \{ \langle (1^r, d_x), i, b \rangle : U^{d_x}(i, b) \text{ accepts in } 2^r \text{ steps} \}$. (It is not hard to show that A is complete for $\text{DTime}(2^n)$ under linear-time many-one reductions.) Let x be given and let $\text{Kt}(x) = m$. Thus, there is a description d_x of length m , such that $U^{d_x}(i, b)$ accepts if and only if $x_i = b$ in at most 2^m steps. The following algorithm M (computable by a two-tape Turing machine) accepts the input (i, b) if and only if $x_i = b$ and runs in time $O(m)$ given oracle A and given finite oracle $(1^m, d_x)$. Given (i, b) , the machine M writes the query $q = \langle (1^m, d_x), i, b \rangle$ onto the query tape. If the oracle accepts q , then M accepts; else it rejects. Clearly, given finite oracle $(1^m, d_x)$, the machine M accepts (i, b) if and only if $b = x_i$. The time required for M is $O(m + |d_x| + |q|) = O(m)$. The length of $| \langle (1^m, d_x) |$ is $O(m)$. Thus, $\text{KT}_M^A(x) = O(m)$. By Definition 6, the first part of the theorem follows.

Now we show that $\text{C}(x) = O(\text{KT}^H(x))$. Let $\text{KT}^H(x) = m$. Thus there is a description d_x of length at most m , such that $U^{H,d_x}(i, b)$ accepts in time m if and only if $x_i = b$. Let r be the number of strings in H of length at most m . Note that r can be written using $O(m)$ bits. Let M be an algorithm with an oracle encoding d_x , r , and m that, given (i, b) , enumerates all the r elements of H that have length at most m and then simulates U^{H,d_x} on input (i, b) . It follows that $\text{C}_M(x) = O(m)$, which is sufficient to prove the claim.

It remains only to show that $\text{KT}^H(x) = O((\text{C}(x) + \log |x|) \log(\text{C}(x) + \log |x|))$. Let $\text{C}(x) = m$ and let d_x be a description of x . The set $L = \{ \langle d_x, i, b \rangle : U^{d_x}(i, b) \text{ accepts} \}$ is a c.e. set; thus it is reducible to H in linear time by a trivial reduction f . Hence there is a machine M that, on input (i, b) with oracle d_x , computes $f(d_x, i, b)$ and poses this question to the oracle H . The running time is $O(m + \log |x|)$. The claim follows by Definition 6. \square

In [11], the approach developed here has been extended to provide new measures of resource-bounded Kolmogorov complexity that are polynomially related to branching program size and formula size.

2.3. Sets of random strings. Our attention will be focused on sets containing strings of high complexity, for various measures of complexity. Specifically, we consider the following sets.

DEFINITION 15. For any Kolmogorov complexity measure μ (such as C, Kt, KS, KT, K^t), define $R_\mu = \{x : \mu(x) \geq |x|/2\}$.

The bound of $|x|/2$ in the definition of R_μ is arbitrary; all of our results hold for any reasonable bound in the range $|x|$ to $|x|^\epsilon$ for any positive ϵ . Essentially, apart from the mere fact that strings in R_μ do not have short descriptions of the appropriate type, all we need is that the set R_μ has polynomial density. We refer to the *density* of a language L as the fraction of all strings of length n that belong to L . L is said to have *polynomial density* if it contains at least $2^n/n^k$ strings of each length n , for some k . The set of μ -incompressible strings, i.e., the set of strings x with $\mu(x) \geq |x|$, is well known to have polynomial density for $\mu = C$ [44]. The same holds for $\mu \in \{\text{Kt}, \text{KS}, \text{KT}\}$ by a trivial counting argument: Since each of these measures μ involves a nontrivial additional cost on top of the length of the describing program d , μ -compressible strings of length n are defined by strings d of length less than $n - 1$, of which there are fewer than 2^{n-1} . We refer the reader to [39] for the detailed proofs of our main theorems in their full generality, for arbitrary ϵ .

PROPOSITION 16. The sets R_C , R_{K^t} , R_{K^t} for any t , and R_{KS} all have polynomial density.

Many of our results about Kt and KT complexity carry over to K^t complexity for certain values of t because of the following observation. A similar statement holds for K^t in relationship to C and for KS in relationship to Kt.

PROPOSITION 17. For some constant $c > 1$,

- (i) $C(x) \leq K^t(x)$ for any time bound t ,
- (ii) $K^t(x) < \text{KT}(x)$ for any time bound $t \geq n + c \log n$,
- (iii) $K^t(x) < \text{Kt}(x)$ for any time bound $t \geq (\log n)^c 2^n$, and
- (iv) $\text{Kt}(x) \leq c \text{KS}(x)$.

Proof. The first item follows directly from the definition. For the second statement, let $m = \text{KT}(x)$. Thus, there is a description d of length less than m such that each bit of x can be produced in time at most m . By Proposition 13, $m \leq n + c \log n$. Thus, the same description d shows that $K^t(x) < m$ for any $t \geq n + c \log n$. The proof for Kt is analogous. The last inequality follows from the fact that if a halting machine runs in time t and space s then $t \leq 2^{O(s)}$. \square

COROLLARY 18. For some constant c ,

- (i) $R_C \subseteq R_{K^t}$ for any time bound t ,
- (ii) $R_{K^t} \subseteq R_{\text{KT}}$ for $t \geq n + c \log n$, and
- (iii) $R_{K^t} \subseteq R_{\text{Kt}}$ for $t \geq n^c 2^n$.

PROPOSITION 19. If $\text{Kt}(x) \leq |x|^\epsilon$, then $K^{2^{n^\epsilon}}(x) \leq |x|^\epsilon$.

The following upper bounds on the complexity of the various sets of random strings are straightforward.

PROPOSITION 20. $R_C \in \text{co-c.e.}$, $R_{K^t} \in \text{E}$, $R_{KS} \in \text{DSPACE}(n)$, and $R_{\text{KT}} \in \text{coNP}$. Also, $R_{K^t} \in \text{EXP}$ for any constructible $t = 2^{n^{O(1)}}$, and $R_{K^t} \in \text{coNP}$ for any constructible $t = n^{O(1)}$.

We show in Corollary 32 that every c.e. set is reducible to R_C via a reduction computable by polynomial size circuits. It follows that R_C requires circuits of size at

least 2^{n^ϵ} for some $\epsilon > 0$. For the rest of these sets R_μ , the best unconditional circuit lower bound known is that none of these sets are in AC^0 .

THEOREM 21. *Let A be any set in AC^0 that contains at least $2^n/n^k$ strings of each length n . Then there is a constant c such that, for every length n , there is a string $x \in A \cap \{0, 1\}^n$ such that $KT(x) \leq \log^c n$.*

Proof. Nisan [48] showed that the Nisan–Wigderson generator can be used to derandomize any probabilistic AC^0 circuit. His construction presents a generator G that takes a seed of length $\log^c n$ and produces pseudorandom output of length n , with the property that any AC^0 circuit C of size n^l and depth d that accepts at least $2^n/n^k$ inputs of length n must accept some string in the range of G . (The constant c depends on the constants l, k , and d .) Furthermore, it was shown in [59] that there is a circuit (actually, even a formula) of size $\log^{O(1)} n$ that takes s and i as input and produces the i th bit of $G(s)$. It follows that, for any seed s , the pseudorandom string $G(s)$ has polylogarithmic KT complexity. \square

COROLLARY 22. *For $t = \Omega(n^2)$, none of the sets R_{Kt} , R_{Kt^t} , R_{KT} , and R_{KS} are in AC^0 .*

2.4. Nonhardness results for R_μ . As stated in the introduction, earlier results had indicated that sets of strings with high resource-bounded Kolmogorov complexity would *not* be complete for the complexity classes in which they reside. In this subsection, we present a few of these nonhardness results as they relate to the sets R_{KT} , R_{KS} , and R_{Kt} .

THEOREM 23.

- (i) R_{Kt} is not hard for EXP under \leq_{tt}^P reductions.
- (ii) R_{KS} is not hard for PSPACE under \leq_T^{\log} reductions.

Proof. We present the proof for R_{KS} . The proof for R_{Kt} is analogous. (The interested reader can consult [6].) Note first that \leq_T^{\log} reducibility coincides with \leq_{tt}^{\log} reducibility [42].

Let T be a subset of $\{0\}^*$ that is in PSPACE but not in L. Suppose $T \leq_{tt}^{\log} R_{KS}$, and let f be the query generator of such a reduction. Note that $KS(f(0^n)) = O(\log n)$. Thus each of the strings y that the reduction queries on input 0^n has $KS(y) = O(\log n)$. Hence, the only strings y for which the query can receive a “yes” answer are of length $O(\log n)$, and for such queries the answer is computable directly in space $O(\log n)$. Hence all of the queries can be answered in space $O(\log n)$ and it follows that $T \in L$, contrary to our choice of T . \square

An interesting picture emerges if we consider even more restrictive reducibilities than polynomial-time and logspace reductions. Most natural examples of NP-complete sets are complete even under many-one reductions computed by uniform AC^0 circuits. However, as the following theorem shows, uniform AC^0 reductions cannot reduce even some very small complexity classes to R_{KS} and R_{Kt} .

THEOREM 24. *R_{Kt} and R_{KS} are not hard for TC^0 under AC^0 many-one reductions.*

Proof. Let A be any set that is hard for TC^0 under AC^0 many-one reductions. Agrawal shows in [1] that A is also hard for TC^0 under length-increasing AC^0 reductions. Thus, there is a length-increasing AC^0 reduction f reducing 0^* to A . Since this reduction f is computable in logspace, it follows that $KS(f(0^n))$ and $Kt(f(0^n))$ are each $O(\log n)$. Since f is length-increasing, it follows that A must contain infinitely many strings of low KS and Kt complexity. \square

Unfortunately, we do *not* know how to argue that the strings $f(0^n)$ in the preceding proof have low KT complexity; in fact it follows easily from Lemma 25 below

that this happens if and only if every problem in the polynomial-time hierarchy has circuits of size $2^{n^{o(1)}}$.

The *linear-time hierarchy* is a subclass of the polynomial-time hierarchy consisting of the union of the classes Σ_k -time(n).

LEMMA 25. *Let s be a monotone nondecreasing function. Every problem in the linear-time hierarchy has circuits of size $s(n)^{O(1)}$ if and only if for each function f computable in AC^0 , $KT(f(0^n)) \leq s(\log n)^{O(1)}$.*

Proof. Suppose that every problem in the linear-time hierarchy has circuits of size $s(n)^{O(1)}$. Let f be computable in AC^0 . By a standard padding argument (see, for instance, [10]) it follows that the set $\{(n, i) : \text{the } i\text{th bit of } f(0^n) \text{ is } 1\}$ is in the linear-time hierarchy. It follows from our hypothesis that this set has circuits of size $s(\log n)^{O(1)}$, and therefore by Theorem 11 that $KT(f(0^n)) \leq s(\log n)^{O(1)}$.

For the other direction, suppose that each function f computable in AC^0 has the property that $KT(f(0^n)) \leq s(\log n)^{O(1)}$. Let A be any problem in the linear-time hierarchy, and consider the function f that maps 0^n for $n = 2^m$ to the characteristic string of A for inputs of length m . The function f is computable in AC^0 . It follows from the hypothesis and Theorem 11 that A on inputs of size m has circuits of size $s(\log 2^m)^{O(1)} = s(m)^{O(1)}$. \square

In particular, Lemma 25 states that SAT has polynomial-size circuits if and only if for every function f computable in uniform AC^0 , $KT(f(0^n))$ is polylogarithmic.

Based on Lemma 25, the best we can offer for R_{KT} is the following conditional nonhardness theorem. The hypothesis to this theorem is unlikely; thus the proper way to interpret this result is that it is unlikely that one will be able to prove that R_{KT} is hard for TC^0 under AC^0 reductions.

THEOREM 26. *If every problem in the polynomial-time hierarchy has circuits of size $2^{n^{o(1)}}$, then R_{KT} is not hard for TC^0 under AC^0 many-one reductions.²*

Proof. Let A and f be as in the proof of Theorem 24. Since f is computable in AC^0 , the hypothesis and Lemma 25 imply that $KT(f(0^n)) \leq n^{o(1)}$. Since f is length-increasing, this means that A has to contain infinitely many strings of low KT complexity. \square

On the other hand, if SAT requires large circuits, the proof of Lemma 25 ensures the existence of an AC^0 computable function f such that $f(0^n)$ has high KT complexity (for every large n that is a power of 2). By concatenating elements of SAT with $f(0^n)$, we obtain a set SAT' that is AC^0 -isomorphic to SAT but where each element of SAT' has large KT complexity. Combining the two directions, we see that SAT requiring large circuits is in fact equivalent to the existence of sets AC^0 -isomorphic to SAT containing only strings of high KT complexity.

The results of this section are related to the issue that was raised by Kabanets and Cai [36] in considering the question of whether or not MCSP is NP-complete under length-increasing reductions. They observed that if MCSP (or R_{KT}) is complete for coNP under length-increasing reductions, then there is a polynomial-time computable length-increasing function f such that each string $f(0^n)$ has high KT complexity, which implies that there is an efficient way to construct the truth tables of certain functions on $m = \Theta(\log n)$ bits that require circuits of size $2^{\epsilon m}$ for some $\epsilon > 0$. As Kabanets and Cai discuss (see also [6]), this implies $BPP = P$.

²In [6], it is asserted that the conclusion of this theorem follows from the weaker assumption that SAT has circuits of size $2^{n^{o(1)}}$. We do not know how to prove this stronger statement.

3. Complexity of R_{Kt} , R_{KS} , and R_{C} . Improving greatly the results in [21], we show that strings of high Kolmogorov complexity are very useful as oracles. Our main technique is to use relativizing hardness-randomness tradeoffs in the contrapositive. In particular we will argue that an appropriate set R_μ of Kolmogorov random strings can be used to distinguish the output of a pseudorandom generator G_f (based on a function f) from truly random strings. This in turn will enable us to efficiently reduce f to R_μ . In this section, we will exploit pseudorandom generators G_f that may take more time to compute than the randomized procedure they try to fool. They yield our hardness results for R_{Kt} , R_{KS} , and R_{C} . In the next section, we will use pseudorandom generators G_f that are more efficiently computable and obtain hardness results for R_{KT} .

We first describe the derandomization tools we will use. Then we present some hardness results in terms of nonuniform (P/poly) reductions that apply to many complexity classes. Finally, we consider some special cases where we are able to strengthen our results to provide uniform reductions.

3.1. Tools. It will be useful to recall the definition of PSPACE-robustness [14]. A language A is PSPACE-robust if $\text{PSPACE}^A = \text{P}^A$. (Here we assume that machines are allowed to ask oracle queries of only polynomial size.) The complete sets for many large complexity classes like PSPACE, EXP, and EXPSPACE, have this property, as well as the complete sets (under linear-time reductions) for classes like DSPACE(n) and E. It will be useful for us to observe that the halting problem is also PSPACE-robust.

THEOREM 27. *The halting problem H is PSPACE-robust.*

Proof. Let M be a PSPACE machine using space n^k . The set $A = \{(1^n, i) : \text{there are at least } i \text{ strings in } H \text{ of length at most } n\}$ is c.e., and thus it is reducible in linear time to H . Using binary search on calls to A , we can compute the number of strings in H of length at most n^k . Now consider the set $B = \{(M, x, r) : M^{H'}(x) \text{ accepts, where } H' \text{ is the set consisting of the first } r \text{ strings of length at most } |x|^k \text{ that show up in the enumeration of } H\}$. Deciding whether M^H accepts x boils down to computing the desired number r (using calls to A) and then making a single call to B . Since both A and B are efficiently reducible to H , the proof is complete. \square

We will use several related constructions that build a pseudorandom generator G_f out of a function f . They are all based on the Nisan–Wigderson paradigm [49]. The authors of [14] construct, for any $\epsilon > 0$, a variant $G_f^{\text{BFNW}} : \{0, 1\}^{n^\epsilon} \mapsto \{0, 1\}^n$ such that for any x of size n^ϵ , the function $G_f^{\text{BFNW}}(x)$ is computable in space $O(n^\epsilon)$ given access to the Boolean function f on inputs of size at most n^ϵ . Moreover, if f is PSPACE-robust, then there is a constant c independent of ϵ , such that each bit of $G_f^{\text{BFNW}}(x)$ is computable in time $n^{\epsilon c}$ with oracle access to f . The following hardness versus randomness tradeoff holds.

THEOREM 28 (see [14, 37]). *Let f be a Boolean function, let $\epsilon > 0$, and denote the pseudorandom generator described above as $G_f^{\text{BFNW}} : \{0, 1\}^{n^\epsilon} \mapsto \{0, 1\}^n$. Let T be a set and $p(n)$ a polynomial. If $|\Pr_{r \in U_n}[r \in T] - \Pr_{x \in U_{n^\epsilon}}[G_f^{\text{BFNW}}(x) \in T]| \geq 1/p(n)$ for all large n , then there exists a polynomial size oracle circuit family $\{C_n\}_{n \in \mathbb{N}}$ with oracle T that computes f and queries T nonadaptively.*

In fact, as observed in [11], close analysis of [14, 37] reveals that the circuit family $\{C_n\}$ can be constructed to be in TC^0 . Thus, in particular, we can also conclude that f is in L^T/poly .

In [34], Impagliazzo and Wigderson reexamine the approach of [14]. For any $\epsilon > 0$, their pseudorandom generator $G_f^{\text{IW98}} : \{0, 1\}^{n^\epsilon} \mapsto \{0, 1\}^n$ is also computable in space

$O(n^\epsilon)$ with oracle access to f on inputs of size at most n^ϵ . It satisfies the following uniform version of Theorem 28. Recall that a function f is *random self-reducible* if there exists a randomized polynomial-time reduction from f to itself such that each individual query the reduction makes on an input of length n is uniformly distributed among the inputs of length n . A function f is *downward self-reducible* if there exists a polynomial-time reduction from f to itself that makes only queries of length strictly less than the input.

THEOREM 29 (see [34]). *Let $f : \{0, 1\}^* \mapsto \{0, 1\}^*$ be a random and downward self-reducible function, $\epsilon > 0$, and $G_f^{\text{IW98}} : \{0, 1\}^{n^\epsilon} \mapsto \{0, 1\}^n$ be the pseudorandom generator described above. Let T be a set and $p(n)$ be a polynomial. If for all large enough n , $|\Pr_{r \in U_n}[r \in T] - \Pr_{x \in U_{n^\epsilon}}[G_f^{\text{IW98}}(x) \in T]| \geq 1/p(n)$, then there exists a probabilistic, polynomial-time Turing machine with oracle T that on input x outputs $f(x)$ with probability at least $2/3$.*

The preceding two theorems provide the key derandomization techniques that are required to prove our completeness results. They are stated in the contrapositive of their original formulations since that is the way we will use them. However, some of our completeness results (namely for EXP and PSPACE) involve uniform reductions that make use of randomness. These reductions can then be further derandomized by applying hardness versus randomness tradeoffs in the standard way. We will make use of the strengthening of the results of [14], as provided by Impagliazzo and Wigderson [33] (see also [55]). Given access to a Boolean function f , they show how to construct a pseudorandom generator $G_f^{\text{IW97}} : \{0, 1\}^{O(\log n)} \mapsto \{0, 1\}^n$ with the following property (see also [37]).

THEOREM 30 (see [33, 37]). *For any $\epsilon > 0$, there exist constants $c, c' > 0$ such that the following holds. Let A be a set and $n > 1$ be an integer. Let $f : \{0, 1\}^{c \log n} \mapsto \{0, 1\}$ be a Boolean function that cannot be computed by oracle circuits of size $n^{c\epsilon}$ with oracle A . Then $G_f^{\text{IW97}} : \{0, 1\}^{c' \log n} \mapsto \{0, 1\}^n$ satisfies the following for any oracle circuit C^A of size at most n : $|\Pr_{r \in U_n}[C^A(r) = 1] - \Pr_{x \in U_{c' \log n}}[C^A(G_f^{\text{IW97}}(x)) = 1]| < 1/n$.*

For x of size $c' \log n$, $G_f^{\text{IW97}}(x)$ is computable in time polynomial in n given access to f on inputs of length $c \log n$.

3.2. Nonuniform hardness results. Our first result illustrates our main technique. Recall that a language L has polynomial density if it contains at least $2^n/n^k$ strings of each length n , for some k . We use the notation $A \leq_{\text{tt}}^{\text{P/poly}} B$ to denote that there exists a truth-table (i.e., nonadaptive) reduction from A to B that is computable by a family of polynomial-size circuits.

THEOREM 31. *Let A be any PSPACE-robust set. Let L be a set of polynomial density such that for every $x \in L$, $\text{KT}^A(x) > |x|^\gamma$ for some constant $\gamma > 0$. Then A is reducible to L via $\leq_{\text{tt}}^{\text{P/poly}}$ reductions.*

Proof. Let f be the characteristic function of A . Consider the generator $G_f^{\text{BFNW}} : \{0, 1\}^{n^\epsilon} \mapsto \{0, 1\}^n$, where we choose ϵ as follows. We know that every bit of G_f^{BFNW} is computable in time $n^{c\epsilon}$ for some constant c independent of ϵ , given access to A . We may assume that $c > 1$, so we set $\epsilon = \gamma/2c$. We claim that any string in the range of G_f^{BFNW} has small KT^A complexity. Let $y = G_f^{\text{BFNW}}(x)$ for some $x \in \{0, 1\}^{n^\epsilon}$. On input x , every bit of $G_f^{\text{BFNW}}(x)$ is computable in time $n^{\gamma/2}$ with access to oracle A . Hence, $\text{KT}^A(y) \leq |x| + O(n^{\gamma/2} \log n) + O(1) \leq n^\gamma$. It follows that L distinguishes the output of G_f^{BFNW} from random, so by Theorem 28, f is $\leq_{\text{tt}}^{\text{P/poly}}$ reducible to L . \square

By Theorem 14 and Proposition 16, we can apply Theorem 31 to the following choices for the pair (A, L) :

- (i) $(A, L) = (\text{the set } A \text{ from Theorem 14, } R_{Kt}),$
- (ii) $(A, L) = (\text{the set } B \text{ from Theorem 14, } R_{KS}),$ and
- (iii) $(A, L) = (H, R_C).$

(In order to apply Theorem 31 we require that A is PSPACE-robust. By Theorem 27, H is PSPACE-robust, and the sets A and B from Theorem 14 are clearly PSPACE-robust, since they are complete for EXP and PSPACE, respectively.) Together with Proposition 20, this leads to the following results.

COROLLARY 32.

- (i) R_{Kt} is complete for EXP under $\leq_{tt}^{P/poly}$ reductions.
- (ii) R_{Kt} is complete for EXP under $\leq_{tt}^{P/poly}$ reductions for any constructible time bound t such that $2^{n^\epsilon} \leq t \leq 2^{n^{O(1)}}$ for some $\epsilon > 0$.
- (iii) R_{KS} is complete for PSPACE under $\leq_{tt}^{P/poly}$ reductions.
- (iv) $H \leq_{tt}^{P/poly} R_C.$

Proof. The results for R_{Kt} , R_{KS} , and R_C are immediate. The result for R_{Kt} follows from Proposition 19 and Theorem 31 (letting A be complete for E under linear-time reductions). \square

The last item in Corollary 32 appears to be the first “efficient” reduction from the halting problem to the Kolmogorov random strings. This should be contrasted with the uniform but very high-complexity truth-table reduction of [40].

A complete set for exponential space can be defined by considering a variant on KS complexity (which could naturally be denoted Ks), where the value to be minimized is $|d| + \log s$. Similarly, complete sets for doubly exponential time can be defined by minimizing $|d| + \log \log t$. This can be generalized to higher complexity classes in the straightforward manner.

It is natural to wonder how much nonuniform advice is needed for the reductions of Corollary 32. For instance, can the reductions be done with linear advice? For the case of deterministic nonadaptive reductions to R_{Kt} , Ronneburger shows in [52] that there is no fixed k such that advice of length n^k is sufficient, even if the reduction is not restricted to run in polynomial time, but instead can run for time 2^{n^k} . In contrast, in the next subsection we consider settings in which no nonuniform advice is needed at all.

3.3. Uniform hardness results. For the special cases of EXP and PSPACE, we are able to show hardness results under uniform notions of reducibility. First we state and prove a general uniform hardness result for PSPACE.

THEOREM 33. *Let L be a set of polynomial density such that for every $x \in L$, $KS(x) > |x|^\gamma$ for some constant $\gamma > 0$. Then $PSPACE \subseteq BPP^L$.*

Proof. We make use of the uniform derandomization technique of [34], together with the following theorem of [57].

THEOREM 34 (see [57]). *There exists a problem in $DSPACE(n)$ that is hard for PSPACE, random self-reducible, and downward self-reducible.*

Let $f \in DSPACE(n)$ be a function that is downward and random self-reducible and that is hard for PSPACE, as guaranteed by Theorem 34. Consider the generator $G_f^{IW98} : \{0, 1\}^{n^{\gamma/2}} \mapsto \{0, 1\}^n$. Since the function G_f^{IW98} is computable in space $O(n^{\gamma/2})$ with oracle access to f on inputs of length $O(n^{\gamma/2})$ and f is computable in linear space, we can compute $G_f^{IW98}(z)$ for $z \in \{0, 1\}^{n^{\gamma/2}}$ in space $O(n^{\gamma/2})$. It follows that every y in the range of G_f^{IW98} satisfies $KS(y) \leq O(|y|^{\gamma/2})$. Since L is a set of at

least polynomial density that contains only strings y of KS complexity at least $|y|^\gamma$, the set L distinguishes the output of G_f^{IW98} from random strings and by Theorem 29, there is a probabilistic procedure with oracle L that on input x outputs $f(x)$ with probability at least $2/3$. Hence, $PSPACE \subseteq BPP^L$. \square

For many sets L satisfying the hypothesis of Theorem 33 (including any such set that lies in $PSPACE/poly$, and all of the sets R_μ for which nonuniform hardness results were presented in the previous section), the BPP reduction of Theorem 33 can be improved to obtain a ZPP reduction. This is a consequence of the next lemma, which shows how probabilistic complexity classes can be derandomized using oracle access to strings of high Kolmogorov complexity. The proof makes use of the by now familiar fact that having access to the truth-table of a hard function can be used to derandomize BPP [49, 33].

LEMMA 35. *Let A be any oracle and L be a set such that $L \in P^A/poly$ and for every $x \in L$, $KT^A(x) > |x|^\gamma$ for some constant $\gamma > 0$.*

(i) $MA^L = NP^L$ if L contains at least one string of almost every length.

(ii) $BPP^L = ZPP^L$ if L is of at least polynomial density.

(iii) $BPP^L = P^L$ if there exists a polynomial time machine with oracle access to L that, on input 0^n , produces an element of L of length n .

All three statements hold even if the respective hypothesis on L holds only for infinitely many constants c and almost all lengths of the form n^c .

Proof. (i) The NP-machine M guesses a string $\chi_f \in L$ of length m , for $m = n^c$, and interprets it as the truth-table of a Boolean function f on inputs of size $\log m$. Since $\chi_f \in L$, $KT^A(\chi_f) \geq m^\gamma$. Thus by Theorem 11, f requires circuits of size at least $\Omega(m^{\gamma/3})$ even when the circuit has access to the oracle A . If, instead of A , the circuit has access to L , then (because of our assumption that L is reducible to A by circuits of size m^k for some $k \geq 1$), the size required to compute f is at least $\Omega(m^{\gamma/6k})$. The function f is of sufficient hardness to construct the generator G_f^{IW97} , as stated in Theorem 30, to stretch $O(\log n)$ random bits into n pseudorandom bits that are indistinguishable from random by any oracle circuit of size n with access to L . Thus we can fully derandomize the probabilistic part of the MA computation.

(ii) The proof is almost identical to the preceding argument. We have the additional assumption that L has polynomial density. Thus, instead of “guessing” the string χ_f as in the preceding argument, we can repeatedly pick strings at random. The expected running time until we find a $\chi_f \in L$ is polynomial. The result follows.

(iii) Instead of randomly choosing a candidate for χ_f as in the proof of statement (ii), we use the generating algorithm from the hypothesis to produce it. The rest of the argument is unchanged. \square

By Theorem 14 and Propositions 16 and 20, the common hypothesis of Lemma 35 holds for $L = R_{Kt}$ (using the set A from Theorem 14 as the set A), for $L = R_{KS}$ (using the set B from Theorem 14 as the set A), and for $L = R_C$ (using $A = H$). Thus one can obtain, e.g., $BPP^{R_{Kt}} = ZPP^{R_{Kt}}$, $BPP^{R_{KS}} = ZPP^{R_{KS}}$, $NP^{R_{Kt}} = MA^{R_{Kt}}$. Moreover, we have the following corollary.

COROLLARY 36. $PSPACE = ZPP^{R_{KS}} \subseteq ZPP^{R_{Kt}}$.

The techniques of Theorem 33 cannot be used to prove hardness results for classes larger than $PSPACE$. For EXP , however, we are able to use the theory of interactive proofs to obtain hardness results under NP-Turing reducibility. First, we mention a result that is implicit in [14].

THEOREM 37 (see [14]). *If $EXP \leq_T^{P/poly} L$, then $EXP \subseteq MA^L$.*

Proof. Let A be any language in EXP . Then A is accepted by a 2-prover interactive proof system with provers computable in EXP [13]. Thus these strategies are

computable in P^L/poly . The MA^L protocol is as follows. Merlin sends Arthur the polynomial-size circuits that, when given access to the oracle L , compute the answers given by the two provers for A . Arthur then executes the MIP protocol, simulating the provers' answers by executing the circuits and querying the oracle L . \square

COROLLARY 38. *Let L be a set of polynomial density in EXP such that for every $x \in L$, $\text{Kt}(x) > |x|^\gamma$ for some constant $\gamma > 0$. Then $\text{EXP} = \text{NP}^L$.*

Proof. The inclusion $\text{NP}^L \subseteq \text{EXP}$ is trivial. For the other inclusion, observe that Corollary 32 implies that the hypothesis to Theorem 37 is satisfied. The theorem now follows from the first item of Lemma 35. \square

COROLLARY 39. $\text{EXP} = \text{NP}^{R_{\text{Kt}}}$.

We also obtain the following curious corollary.

COROLLARY 40. $P^{R_{\text{Kt}}} \neq \text{NP}^{R_{\text{Kt}}} = \text{EXP}$ for any constructible monotone t such that $n^{O(1)}2^{nt}(n-1) \leq t(n) \leq 2^{n^{O(1)}}$, e.g., $t = 2^{n^2}$.

Proof. The equality follows from Corollary 38 by the third part of Proposition 17 and by Proposition 20. The inequality holds because the proof in [20] showing that R^t is not complete for EXP under \leq_{T}^P reductions carries over unchanged to R_{Kt}^t . We include a somewhat simplified argument.

Consider a \leq_{T}^P reduction from a unary language A to R_{Kt}^t . We claim that for inputs $x = 0^n$ of sufficiently large length, none of the queries of length n or more can be in R_{Kt}^t . Otherwise, we could describe the first such query q by n and the index i of the query. This yields a description of length $O(\log n) < n$. Moreover, we can generate q from this description in time $n^{O(1)}2^{nt}(n-1) \leq t(n)$ by simulating the polynomial-time reduction up to the moment it generates its i th query and answering all earlier queries by running the trivial exponential-time algorithm for R_{Kt}^t . This contradicts the hypothesis that q is in R_{Kt}^t . Thus, we can decide A as follows: Run the reduction by answering queries of length n or more negatively and running the trivial exponential-time algorithm for R_{Kt}^t on smaller queries. This yields a correct algorithm for A that runs in $\text{DTime}(2^{dn})$ for some fixed constant independent of A . However, the time hierarchy theorem implies that there are unary languages A in EXP that have higher complexity than this. \square

To the best of our knowledge, all prior examples of oracles for which $P \neq \text{NP}$ have been specifically constructed for that purpose. We know of no other “natural” examples of sets A for which $P^A \neq \text{NP}^A$.

We are unable to present any completeness or hardness results under deterministic polynomial-time reductions for any of the resource-bounded notions of Kolmogorov complexity that we have studied. It is interesting to observe, however, that we are able to do better when we consider the undecidable set R_C .

THEOREM 41. $\text{PSPACE} \subseteq P^{R_C}$.

Theorem 41 follows immediately from Theorem 33 and the following lemma, which implies that $\text{BPP}^{R_C} = P^{R_C}$ by the third item of Lemma 35 applied to $A = H$ and $L = R_C$. It is observed in [8] that very similar techniques suffice to show that $\text{NEXP} \subseteq \text{NP}^{R_C}$.

LEMMA 42 (see [19]). *There is a polynomial-time algorithm with oracle access to R_C that, on input 0^n , outputs a string of length n in R_C .*

We include a simple proof of Lemma 42 which works for our standard version of R_C , i.e., with a randomness threshold of $|x|/2$. Our proof also works for the smaller randomness thresholds we consider but does not work for a randomness threshold close to $|x|$. [19] gives a more complicated proof for the case where the oracle used is $\{x : C(x) \geq |x|\}$, i.e., for a randomness threshold of $|x|$; that proof works for any

reasonable randomness threshold between $|x|$ and $|x|^\epsilon$ for any positive ϵ (see Remark 2 after Theorem 10 in [19]).

Proof of Lemma 42. (We prove the lemma for the standard randomness threshold of $|x|/2$.) For any $m = n^{O(1)}$, we show how to construct a string z of length m such that $C(z) \geq |z|/2$ via a polynomial-time computation with access to oracle R_C . We will use the following property referred to as symmetry of information in [44, section 2.8]: There exists a constant c_1 such that for any strings z and y , $C(zy) \geq C(z) + C(y|z) - c_1 \log |zy|$.

We build z inductively. We start with z equal to the empty string. Assume (inductively) that $C(z) \geq |z|/2$. Try all strings y of length $2c_1 \log m$, and use the oracle R_C to see if $C(zy) \geq |zy|/2$. We are guaranteed to find such a y , since $C(y|z) \geq |y|$ holds for most y , and for any such y , $C(zy) \geq C(z) + C(y|z) - c_1 \log |zy| \geq |z|/2 + |y| - c_1 \log |zy| \geq |zy|/2$. Set z to be zy . \square

In a similar way, part 3 of Lemma 35 would allow us to obtain a *deterministic* polynomial-time reduction of PSPACE to R_{KS} in Corollary 36 if there was a deterministic method to construct a string of high KS complexity using R_{KS} as an oracle. We do not know if this is possible.

The question of whether or not long elements of R_{Kt} can be obtained in polynomial time relative to R_{Kt} turns out to be of critical importance in understanding the completeness properties of R_{Kt} . Let us formalize this as follows. Let Q denote the proposition “For all c there is a function f computable in polynomial time relative to R_{Kt} such that $f(0^n) \in R_{Kt} \cap \{0, 1\}^{n^c}$.” If Q is true, then Corollary 36 and the third item of Lemma 35 imply that $PSPACE \subseteq ZPP^{R_{Kt}} = P^{R_{Kt}}$; moreover, $R_{Kt} \notin P$ since otherwise Corollary 39 implies that $EXP = NP^{R_{Kt}} = NP \subseteq PSPACE \subseteq P^{R_{Kt}} = P$. On the other hand, if Q is false then $EXP \neq P^{R_{Kt}}$. This follows from an argument similar to one presented in the proof of Corollary 40: If there is some c for which no computation in $P^{R_{Kt}}$ on input 0^n can access a string in R_{Kt} of length greater than n^c , then it follows that every unary language in $P^{R_{Kt}}$ is in $DTime(2^{3n^c})$, which contradicts the time hierarchy theorem.

It should be noted in this regard that the property of symmetry of information, which was used in the proof of Lemma 42, provably does not hold for R_{Kt} [52].

Let us mention one other consequence of our PSPACE-completeness result. In [62] Watanabe and Tang showed that \leq_T^P and \leq_m^P reducibilities yield different classes of PSPACE-complete sets if and only if \leq_T^{BPP} and \leq_m^P reducibilities yield different PSPACE-complete sets. Using similar techniques, we are able to prove the following.

THEOREM 43. *There is a tally set T such that $PSPACE = P^{R_{KS} \cup T}$, where at least one of the following holds:*

- (i) R_{KS} is complete for PSPACE under \leq_{tt}^P reductions, but not under \leq_T^{\log} reductions, or
- (ii) $R_{KS} \cup T$ is complete for PSPACE under \leq_T^P reductions, but not under \leq_m^P reductions.

Proof. By Theorem 33 there is a probabilistic Turing machine M running in time n^k accepting QBF with oracle R_{KS} , with error probability less than 2^{-2n} . Let S be the set of all strings r such that r is the lexicographically first string of length n^k with the property that, for all strings x of length n , $M^{R_{KS}}(x, r)$ accepts if and only if $x \in QBF$. The set S is in PSPACE and, for almost all n , S contains exactly one string of length n^k . We encode the “good” coin flip sequences of S in the tally set $T = \{0^{2n^k+i} : \text{the } i\text{th bit of the unique string of length } n^k \text{ in } S \text{ is } 1\}$. The set R_{KS} contains only a finite number of strings in 0^* . It is immediate that $T \in PSPACE$ and that $QBF \in P^{R_{KS} \cup T}$.

If $R_{KS} \cup T$ is not complete for PSPACE under \leq_m^P reductions, then the second condition of the theorem holds.

On the other hand, if $QBF \leq_m^P R_{KS} \cup T$, then we can apply a standard argument of [17]. Berman showed that if a length-decreasing self-reducible set A is \leq_m^P -reducible to a tally set, then $A \in P$. A similar argument applied to the \leq_m^P reduction from $A = QBF$ to $R_{KS} \cup T$ yields a \leq_{tt}^P reduction from QBF to R_{KS} . We sketch the argument below for completeness. This, combined with Theorem 23, shows that the first condition of the theorem holds.

The modification of Berman's argument goes as follows. Let M denote the length-decreasing self-reduction of A , and f the reduction from A to $R_{KS} \cup T$. We build a reduction tree starting from the given input x as the single node. In the k th phase, we apply M to the active leaves of length $|x| + 1 - k$. Those nodes that map under f to a string outside of 0^* are made inactive. For each integer $m \geq 0$, we choose a representative among all nodes (if any) which f maps to 0^m . We keep the representative active and make all the other nodes which f maps to 0^m inactive. This process finishes after at most $|x|$ phases. Since the number of active nodes is polynomially bounded, each step involves only a polynomial amount of work. The leaves which f maps to strings outside of 0^* constitute the truth-table queries we make to R_{KS} . Once we know the answers to these queries, we can determine the answers to all remaining nodes in the reduction tree from bottom to top. \square

Note that in either case, we obtain a set that is \leq_T^P -complete but not \leq_m^{\log} -complete for PSPACE, namely, R_{KS} or $R_{KS} \cup T$. It was already known (using the techniques of [61]) that \leq_T^P and \leq_m^{\log} reducibilities provide different classes of PSPACE-complete sets, but the preceding theorem provides fairly "natural" examples of sets witnessing the difference.

4. Complexity of R_{KT} . The previous section paints an illuminating picture about the hardness of sets with high Kt, KS, and C complexity for PSPACE, EXP, and larger complexity classes. In this section, we explore what these techniques have to say about the hardness of R_{KT} , a set in coNP. We are not able to show completeness of R_{KT} for coNP, but we can show the hardness of R_{KT} under randomized polynomial-time reductions for problems that are thought to be NP-intermediate: Discrete Log, Factorization, and certain lattice problems.

We point out that the set R_{KT} seems closely related to the set MCSP defined in [36] as $\{(x, s) : \text{SIZE}(x) \leq s\}$. Although we do not know of efficient reductions between R_{KT} and MCSP in either direction, all our hardness results for R_{KT} also hold for MCSP. Based on a connection with natural proofs, [36] showed that if MCSP is in P, then, for any $\epsilon > 0$, there is a randomized algorithm running in time 2^{n^ϵ} that factors Blum integers well on the average. Our results imply that if MCSP is in P, then there is a randomized polynomial-time algorithm that factors arbitrary integers.

4.1. Tools. In order to prove hardness results for R_{KT} , we need to apply hardness versus randomness tradeoffs for pseudorandom generators G_f of lower computational complexity than in section 3. We will use the cryptographic pseudorandom generators that developed out of the seminal work by Blum and Micali [18], and by Yao [63]. In [30], it is shown how to construct such a pseudorandom generator $G_f^{\text{HILL}} : \{0, 1\}^n \mapsto \{0, 1\}^{2n}$ out of any function f . The pseudorandom generator G_f^{HILL} is computable in polynomial time given access to f and is secure provided f is one-way.

The known hardness versus randomness tradeoffs for G_f^{HILL} differ in two relevant respects from those used in section 3. First, breaking G_f^{HILL} only lets us invert f

on a nonnegligible fraction of the inputs, but not necessarily on all inputs. The implicit or explicit random self-reducibility of the problems considered in section 3 allowed us to make the transition from “nonnegligible fractions of the domain” to “the entire domain.” However, unlike for EXP and PSPACE, there are no NP-complete problems that are known to be random self-reducible. In fact, there provably are no nonadaptively random self-reducible NP-complete sets unless the polynomial-time hierarchy collapses [25].

Nevertheless, for some specific NP-intermediate problems h and polynomial-time computable functions f (where h may or may not coincide with f^{-1}), a worst-case to average-case connection is known. That is, inverting f on a nonnegligible fraction of the inputs allows one to compute h efficiently on any input. We are able to prove hardness of R_{KT} for such problems.

The second difference from section 3 is that the uniform hardness versus randomness tradeoffs in [30] are as strong as the nonuniform ones. Therefore, unlike in section 3, we need only to consider uniform reductions here.

We will apply G_f^{HILL} for functions f that take some additional parameters y besides the actual input x . In the case of the Discrete Log problem, for example, y will describe the prime p and the generator g of \mathbb{Z}_p^* defining the basis for the logarithm. More precisely, we will consider functions of the form $f(y, x)$ that are length-preserving for every fixed y , i.e., $f_y : \{0, 1\}^n \mapsto \{0, 1\}^n$, where $f_y(x) = f(y, x)$. The function $f(y, x)$ will be computable uniformly in time polynomial in $|x|$ (so without loss of generality, $|y|$ is polynomially bounded in $|x|$). The hardness versus randomness tradeoff in [30] can be stated as follows.

THEOREM 44 (see [30]). *Let $f(y, x)$ be computable uniformly in time polynomial in $|x|$. For any oracle L , polynomial-time probabilistic oracle Turing machine M , and polynomial p , there exists a polynomial-time probabilistic oracle Turing machine N and polynomial q such that the following holds for any n and y : If*

$$\left| \Pr_{|r|=2n,s} [M^L(y, r, s) = 1] - \Pr_{|x|=n,s} [M^L(y, G_{f_y}^{\text{HILL}}(x), s) = 1] \right| \geq 1/p(n),$$

then

$$\Pr_{|x|=n,s} [f(y, N^L(y, f(y, x), s)) = f(y, x)] \geq 1/q(n),$$

where r and x are chosen uniformly at random and s denotes the internal coin flips of M or N , respectively.

Theorem 44 states that if there exists a distinguisher with access to an oracle L that distinguishes the output of G_f^{HILL} from the uniform distribution, then oracle access to L suffices to invert f on a polynomial fraction of the inputs. We now argue that such a distinguisher exists in the case where L denotes R_{KT} or any set of polynomial density that contains no strings of small KT complexity.

THEOREM 45. *Let L be a language of polynomial density such that, for some $\epsilon > 0$, for every $x \in L$, $KT(x) \geq |x|^\epsilon$. Let $f(y, x)$ be computable uniformly in time polynomial in $|x|$. There exists a polynomial-time probabilistic oracle Turing machine N and polynomial q such that for any n and y*

$$\Pr_{|x|=n,s} [f(y, N^L(y, f(y, x), s)) = f(y, x)] \geq 1/q(n),$$

where x is chosen uniformly at random and s denotes the internal coin flips of N .

Proof. Let $G_y(x)$ denote $G_{f_y}^{\text{HILL}}(x)$. As in [51], we use the construction of [29] to modify G_y and obtain a pseudorandom generator G'_y producing longer output. Specifically, the proof of Theorem 4.1 of [51] constructs $G'_y : \{0, 1\}^n \mapsto \{0, 1\}^{2^k}$, and shows that if $z = G'_y(x)$, then $\text{SIZE}(z) = (n + k)^{O(1)}$. We can pick $k = O(\log n)$ such that for each x and polynomially bounded y , $\text{KT}(G'_y(x)) < |G'_y(x)|^\epsilon$. Thus L is a statistical test that accepts many random strings of length $|x|^{O(1)}$, but rejects all pseudorandom strings. As in [51], this gives us a probabilistic oracle machine M using L that distinguishes G_y from the uniform distribution. We then apply Theorem 44. \square

4.2. Hardness results. We now apply Theorem 45 to obtain our hardness results for R_{KT} . As will be clear from the proofs, R_{KT} can be replaced by any suitably dense language containing no strings of KT-complexity less than n^ϵ for some $\epsilon > 0$, such as R_{KT^t} for polynomial t . Other examples of such sets can be found in P^{MCSP} , and thus our hardness results carry over to MCSP as well. As observed in [11], the reductions in this section show that, under popular cryptographic assumptions, it is impossible even to approximate $\text{KT}(x)$ or $\text{SIZE}(x)$ within $|x|^{1-\epsilon}$.

We first consider the Discrete Log Problem, which takes as input a triple (p, g, z) , where p is a prime number, $0 < g < p$, and $0 < z < p$, and outputs a positive integer i such that $g^i \equiv z \pmod{p}$, or 0 if there is no such i . We have the following theorem.

THEOREM 46. *The Discrete Log Problem is computable in $\text{BPP}^{R_{\text{KT}}}$.*

Proof. On input (p, g, z) , we first check if p is prime [3]. Let n be the number of bits in p , and let y denote the pair (p, g) . Consider the function $f(y, x) = g^x \pmod{p}$. Theorem 45 with $L = R_{\text{KT}}$ gives us a polynomial-time probabilistic oracle Turing machine N such that for some polynomial q and all p and g , with probability at least $1/q(n)$ over randomly chosen x and s , $N^{R_{\text{KT}}}(p, g, g^x, s)$ produces an output i such that $g^i \equiv g^x \pmod{p}$.

Now we make use of the self-reducibility properties of the Discrete Log Problem. In particular, on our input (p, g, z) , we choose many more than $q(n)$ values v at random and run algorithm N on input $(p, g, zg^v \pmod{p})$. If z is in the orbit of g , then with high probability at least one of these trials will return a value u such that $g^u = zg^v \pmod{p}$, which means that we can pick $i = u - v$ and obtain $z \equiv g^i \pmod{p}$. On the other hand, if none of the trials is successful, then with high probability z is not in the orbit of g and the algorithm should return 0. \square

We are not able to improve our reduction from a BPP-reduction to a ZPP-reduction. That is, we know of no analogue of Lemma 35 for KT-complexity. We note that, for inputs (p, g, x) such that x is in the orbit of g , which is the usual class of inputs for which the discrete log is of interest, we do have ZPP-like behavior, since we can check whether the number i we obtain satisfies $g^i \equiv x \pmod{p}$. However, when x is not in the orbit of g , we obtain no *proof* of this fact—merely strong evidence.

The proof of Theorem 46 relies on the random self-reducibility of the Discrete Log Problem. In the terminology of section 4.1, this allows us to consider f^{-1} as the problem h that reduces to R_{KT} . The next problem we consider is a first example where h differs from f^{-1} .

THEOREM 47. *Factorization is computable in $\text{ZPP}^{R_{\text{KT}}}$.*

Proof. Consider Rabin's candidate one-way function $f(y, x) = x^2 \pmod{y}$, where $0 \leq x < y$. Theorem 45 with $L = R_{\text{KT}}$ gives us a probabilistic polynomial-time procedure with oracle access to R_{KT} that, for any fixed y , finds an inverse of $f(y, x)$ for a fraction at least $1/|y|^{O(1)}$ of the x 's with $0 \leq x < y$. Rabin [50] showed how

to use such a procedure and some randomness to efficiently find a nontrivial factor of y in case y is composite. This leads to a $\text{BPP}^{R_{\text{KT}}}$ algorithm for factoring. Since primality is in P [3], we can avoid errors and obtain the promised $\text{ZPP}^{R_{\text{KT}}}$ factoring algorithm. \square

Since Ajtai’s seminal paper [4, 5], several worst-case to average-case connections have been established for lattice problems. We will exploit these next.

We first review some lattice terminology. We refer to [22] for more background. Given a set B of n linearly independent vectors b_1, \dots, b_n over \mathbb{R}^n , the set $L(B) = \{ \sum_{i=1}^n z_i b_i \mid z_i \in \mathbb{Z} \}$ is called the lattice spanned by the basis B . We consider the usual notion of length of a vector v , $|v| = \sqrt{\sum v_i^2}$, and define the length of a set of vectors as the length of a longest vector in that set. For a lattice $L(B)$, $\lambda_i(B)$, $1 \leq i \leq n$, denotes the length of a shortest set of i linearly independent vectors from $L(B)$. The covering radius of $L(B)$, $\rho(B)$, is defined as the smallest ρ such that the union of all spheres of radius ρ centered at the points of $L(B)$ covers the entire space \mathbb{R}^n .

Applying our technique to Ajtai’s worst-case to average-case connections and their subsequent improvements and extensions [24, 47], we obtain the following hardness results for R_{KT} .

THEOREM 48. *For every $\epsilon > 0$, we can solve each of the following problems in $\text{BPP}^{R_{\text{KT}}}$. Given a basis $B \subseteq \mathbb{Z}^n$ (and a vector $t \in \mathbb{Z}^n$), find*

- (i) *(Shortest Independent Vector Problem (SIVP)) a set of n linearly independent vectors in $L(B)$ of length at most $n^{3+\epsilon} \cdot \lambda_n(B)$,*
- (ii) *(Shortest Basis Problem (SBP)) a basis for $L(B)$ of length at most $n^{3.5+\epsilon} \cdot \lambda_n(B)$,*
- (iii) *(Length of Shortest Vector Problem (LSVP)) a value $\tilde{\lambda}$, such that $\lambda_1(B) \leq \tilde{\lambda} \leq \omega(n^{3.5} \log n) \cdot \lambda_1(B)$,*
- (iv) *(Unique Shortest Vector Problem (Unique-SVP)) a shortest vector in $L(B)$ in case $\lambda_2(B) > n^{4+\epsilon} \cdot \lambda_1(B)$,*
- (v) *(Closest Vector Problem (CVP)) a vector $u \in L(B)$ such that $|u - t| \leq n^{3.5+\epsilon} \cdot \lambda_n(B)$,*
- (vi) *(Covering Radius Problem (CRP)) a value $\tilde{\rho}$, s.t. $\rho(B) \leq \tilde{\rho} \leq \omega(n^{2.5} \log n) \cdot \rho(B)$.*

The approximation factors in Theorem 48 represent the current state-of-the-art but are likely to be improved in the future.

In order to prove the statements above, we will use the following worst-case to average-case connection.

THEOREM 49 (see [5, 24]). *For any $\epsilon > 0$, there exist functions $q(n) = n^{O(1)}$ and $m(n) = n \log^2 q$ with q a power of 2 such that the following holds. For every $c > 0$, if there is probabilistic algorithm A , such that, with probability at least $1/n^c$ (over $M \in \mathbb{Z}_q^{n \times m}$ and over the random choices of A), $A(M)$ outputs a nonzero vector $u \in \mathbb{Z}^m$ such that $|u| \leq m$ and $Mu \equiv 0 \pmod q$, then there exists a BPP^A -algorithm A' that, given any basis $B \subseteq \mathbb{Z}^n$, outputs with high probability a set of n linearly independent vectors in $L(B)$ of length at most $n^{3+\epsilon} \cdot \lambda_n(B)$.*

Proof of Theorem 48. Let $q(n)$ and $m(n)$ be as in Theorem 49. Consider the collection of functions $f_{q,m} : \mathbb{Z}_q^{n \times m} \times \{0, 1\}^m \mapsto \mathbb{Z}_q^{n \times m} \times \mathbb{Z}_q^n$ defined by $f_{q,m}(M, v) = (M, Mv \pmod q)$. Letting $y = (q, m)$ and $x = (M, v)$, the function $f_y(x)$ is computable uniformly in time $|x|^{O(1)}$. By Theorem 45, there exists a polynomial-time probabilistic algorithm N with oracle access to R_{KT} that computes a preimage of $f_{q,m}(M, v)$ for at least a $1/n^{O(1)}$ fraction of the inputs (M, v) .

For any fixed M , f maps to at most q^n different values. Thus there can be at

most q^n vectors v , such that $f(M, v) \neq f(M, v')$ for all $v' \neq v$. In other words, for a fraction at least $1 - q^n/2^m > 1 - O(1/2^n)$ of the vectors v , there is a $v' \neq v$ with $f(M, v') = f(M, v)$. Therefore, if we pick M and v uniformly at random and run N to invert $f(M, Mv \bmod q)$, with probability at least $1/n^{O(1)}$ we obtain a vector $v' \in \mathbb{Z}^m$ such that $v' \neq v$ and $Mv \equiv Mv' \bmod q$.

Setting $u = v - v'$ yields a nonzero vector in \mathbb{Z}^m of length $|u| \leq m$ and satisfying $Mu \equiv 0 \bmod q$. This vector u is the output of the algorithm A on input M . Applying Theorem 49 to A yields the $\text{BPP}^{R_{\text{KT}}}$ algorithm for the SIVP.

The reductions for the SBP, the Unique-SVP, and the CVP follow from the one for the SIVP by arguments given in [24, 23]. The results for the LSVP and the CRP are obtained in a similar way based on the variants of the candidate one-way function f and the worst-case to average-case connection corresponding to Theorem 49 presented in [47]. \square

4.3. R_{KT} versus NP. Given our inability to prove that R_{KT} is coNP-complete, one may wonder whether R_{KT} is in NP. If it is, then this would provide a dense combinatorial property in NP that is useful against P/poly, contrary to a conjecture of Rudich [53]. We can also show the following.

THEOREM 50. *If R_{KT} is in NP, then $\text{MA} = \text{NP}$.*

Proof. It is shown in [32] that if an NP machine can, on input of length n , find the truth table of a function of size $n^{O(1)}$ with large circuit complexity, then $\text{MA} = \text{NP}$. Certainly this is easy if R_{KT} is in NP. \square

This observation (similar to ones in [32]) cannot be taken as evidence that $R_{\text{KT}} \notin \text{NP}$, since many conjecture that MA is equal to NP. However, it does show that proving R_{KT} in NP would require nonrelativizing proof techniques.

5. Open problems. In this paper, we present KT as a notion of time-bounded Kolmogorov complexity with close connections to circuit complexity. In so doing, we expose connections between circuit complexity and traditional Kolmogorov complexity and Levin's measure Kt. KT-complexity is also useful as a tool for summarizing some recent progress in the field of derandomization, and for describing the theory of natural proofs from the standpoint of Kolmogorov complexity. For an exposition along these lines, see [6].

The measures KT, KS, and Kt provide natural and interesting examples of apparently intractable problems in NP, PSPACE, and EXP that are not complete under the more familiar notions of reducibility and hence constitute a fundamentally new class of complete problems. It is worth pointing out that variants of these sets (such as $\{x : \text{Kt}(x) \geq |x|/3\}$) appear to be incomparable to our standard $R_{\text{Kt}} = \{x : \text{Kt}(x) \geq |x|/2\}$ under \leq_m^{P} reductions. As another example of this phenomenon, we are unable to present any reduction between $\{x : \text{KT}_U(x) \geq |x|/2\}$ and $\{x : \text{KT}_{U'}(x) \geq |x|/2\}$, where U and U' are two universal machines. This deserves further investigation.

Corollaries 39 and 36 combine to show that $\text{EXP} = \text{ZPP}$ if and only if $R_{\text{Kt}} \in \text{ZPP}$. This is because $R_{\text{Kt}} \in \text{EXP}$ and if $R_{\text{Kt}} \in \text{ZPP}$ then $\text{EXP} = \text{PSPACE} = \text{ZPP}^{R_{\text{Kt}}}$. This suggests that one might be able to show that R_{Kt} is complete under ZPP reductions, but we have not been able to show this.

Can one settle (one way or the other) the question of whether or not R_{Kt} is complete for EXP under $\leq_{\text{T}}^{\text{P}}$ reductions? For instance, is it possible to construct an element of R_{Kt} in $\text{P}^{R_{\text{Kt}}}$ (which would suffice to prove completeness, as in the proof of

Theorem 41)? Or is R_{Kt} more similar to the Buhrman–Mayordomo set R^t (which is provably *not* complete) in this regard?

Is there some sense in which KT complexity yields NP-complete sets? For instance, what can one say about $\{(x, y, i) : \text{KT}(x|y) \leq i\}$? The result of [58] is intriguing in this light; in [58] Vazirani and Vazirani presented a language in NP that is complete under probabilistic reductions that is not known to be complete under deterministic reductions. Their problem superficially seems to be related to time-bounded Kolmogorov complexity.

Other intriguing open questions are: Is $\text{PSPACE} \subseteq \text{P}^{R_{\text{KS}}}$? Is the EXP-complete set R_{Kt} in P? Is it in $\text{AC}^0[\oplus]$? We know of no fundamental obstacles that lie in the way of a direct proof that R_{Kt} is not in P, although it was pointed out to us by Rahul Santhanam that $\text{EXP} = \text{ZPP}$ if and only if there is a set in P of polynomial density containing only strings of Kt-complexity $\geq n^\epsilon$. The backward implication follows via the same argument that we used to show that $\text{EXP} = \text{ZPP}$ if and only if $R_{\text{Kt}} \in \text{ZPP}$. For the other direction, assume that $\text{EXP} = \text{ZPP}$, and let B be a set in EXP of polynomial density such that neither B nor its complement has an infinite subset in $\text{DTime}(2^n)$ [28]. Let A be the set in P consisting of (string, witness) pairs for the ZPP machine accepting B . If A contains infinitely many strings of low Kt complexity, then B contains infinitely many instances where membership can be decided in time 2^n , contrary to our choice of B .

The containment of PSPACE in $\text{P}^{R_{\text{C}}}$ raises the question of whether it might be possible to obtain characterizations of complexity classes in terms of efficient reductions to R_{C} . This is studied in more detail in [8] (where a nontrivial characterization of P having this flavor is presented).

Acknowledgments. We acknowledge illuminating discussions with Lance Fortnow, as well as helpful conversations of the first author with Paul Beame, Henry Cohn, Chris Umans, and Ramarathnam Venkatesan during a visit to Microsoft Research. We thank Jin-Yi Cai and Daniele Micciancio for clarifying some issues regarding lattice problems. We also acknowledge fruitful discussions with Steven Rudich, Valentin Kabanets, Manindra Agrawal, Troy Lee, Neil Jones, Rahul Santhanam, Kolya Vereshchagin, John Hitchcock, and Pavan Aduri. Finally, we thank the anonymous referees for their suggestions.

REFERENCES

- [1] M. AGRAWAL, *The first-order isomorphism theorem*, in FST TCS 2001: Foundations of Software Technology and Theoretical Computer Science, Lecture Notes in Comput. Sci. 2245, Springer-Verlag, Berlin, 2001, pp. 70–82.
- [2] M. AGRAWAL, E. ALLENDER, R. IMPAGLIAZZO, R. PITASSI, AND S. RUDICH, *Reducing the complexity of reductions*, Comput. Complexity, 10 (2001), pp. 117–138.
- [3] M. AGRAWAL, N. KAYAL, AND N. SAXENA, *PRIMES is in P*, Ann. of Math. (2), 160 (2004), pp. 781–793.
- [4] M. AJTAI, *Generating hard instances of lattice problems (extended abstract)*, in Proceedings of the ACM Symposium on Theory of Computing (STOC), ACM, New York, 1996, pp. 99–108.
- [5] M. AJTAI, *Generating hard instances of lattice problems*, in Complexity of Computations and Proofs, J. Krajíček, ed., Quad. Mat. 13, Aracne, Rome, 2004, pp. 1–32.
- [6] E. ALLENDER, *When worlds collide: Derandomization, lower bounds, and Kolmogorov complexity*, in FST TCS 2001: Conference on Foundations of Software Technology and Theoretical Computer Science, Lecture Notes in Comput. Sci. 2245, Springer-Verlag, Berlin, 2001, pp. 1–15.
- [7] E. ALLENDER, *NL-printable sets and nondeterministic Kolmogorov complexity*, Theoret. Comput. Sci., 355 (2006), pp. 127–138.

- [8] E. ALLENDER, H. BUHRMAN, AND M. KOUCKÝ, *What can be efficiently reduced to the Kolmogorov random strings?* Ann. Pure Appl. Logic, 138 (2006), pp. 2–19.
- [9] E. ALLENDER, H. BUHRMAN, M. KOUCKÝ, D. VAN MELKEBEEK, AND D. RONNEBURGER, *Power from random strings*, in Proceedings of the 43rd IEEE Annual Symposium on Foundations of Computer Science, IEEE Computer Society, Los Alamitos, CA, 2002, pp. 669–678.
- [10] E. ALLENDER AND V. GORE, *Rudimentary reductions revisited*, Inform. Process. Lett., 40 (1991), pp. 89–95.
- [11] E. ALLENDER, M. KOUCKÝ, D. RONNEBURGER, AND S. ROY, *Derandomization and distinguishing complexity*, in Proceedings of the 18th IEEE Conference on Computational Complexity, IEEE Computer Society, Los Alamitos, CA, 2003, pp. 209–220.
- [12] L. ANTUNES, L. FORTNOW, AND D. VAN MELKEBEEK, *Computational depth*, in Proceedings of the IEEE Conference on Computational Complexity, IEEE Computer Society, Los Alamitos, CA, 2001, pp. 266–273.
- [13] L. BABAI, L. FORTNOW, AND C. LUND, *Non-deterministic exponential time has two-prover interactive protocols*, Comput. Complexity, 1 (1991), pp. 3–40.
- [14] L. BABAI, L. FORTNOW, N. NISAN, AND A. WIGDERSON, *BPP has subexponential time simulations unless EXPTIME has publishable proofs*, Comput. Complexity, 3 (1993), pp. 307–318.
- [15] D. MIX BARRINGTON, N. IMMERMANN, AND H. STRAUBING, *On uniformity within NC^1* , J. Comput. System Sci., 41 (1990), pp. 274–306.
- [16] A. BEN-AMRAM AND N. JONES, *Computational complexity via programming languages: Constant factors do matter*, Acta Inform., 37 (2000), pp. 83–120.
- [17] P. BERMAN, *Relationship between density and deterministic complexity of NP-complete languages*, in ICALP, Lecture Notes in Comput. Sci. 62, Springer-Verlag, Berlin, 1978, pp. 63–71.
- [18] M. BLUM AND S. MICALI, *How to generate cryptographically strong sequences of pseudo-random bits*, SIAM J. Comput., 13 (1984), pp. 850–864.
- [19] H. BUHRMAN, L. FORTNOW, I. NEWMAN, AND N. VERESHCHAGIN, *Increasing Kolmogorov complexity*, in Proceedings of the Symposium on Theoretical Aspects of Computer Science (STACS), Lecture Notes in Comput. Sci. 3404, Springer-Verlag, Berlin, 2005, pp. 412–421.
- [20] H. BUHRMAN AND E. MAYORDOMO, *An excursion to the Kolmogorov random strings*, J. Comput. System Sci., 54 (1997), pp. 393–399.
- [21] H. BUHRMAN AND L. TORENVLIET, *Randomness is hard*, SIAM J. Comput., 30 (2000), pp. 1485–1501.
- [22] J.-Y. CAI, *Some recent progress on the complexity of lattice problems*, in Proceedings of the IEEE Conference on Computational Complexity, IEEE Computer Society, Los Alamitos, CA, 1999, pp. 158–179.
- [23] J.-Y. CAI, *On the average-case hardness of CVP*, in Proceedings of the IEEE Symposium on Foundations of Computer Science (FOCS), IEEE Computer Society, Los Alamitos, CA, 2001, pp. 308–319.
- [24] J.-Y. CAI AND A. NERURKAR, *An improved worst-case to average-case connection for lattice problems*, in Proceedings of the IEEE Symposium on Foundations of Computer Science (FOCS), IEEE Computer Society, Los Alamitos, CA, 1997, pp. 468–477.
- [25] J. FEIGENBAUM AND L. FORTNOW, *Random-self-reducibility of complete sets*, SIAM J. Comput., 22 (1993), pp. 994–1005.
- [26] M. FÜRER, *The tight deterministic time hierarchy*, in Proceedings of the ACM Symposium on Theory of Computing (STOC), ACM, New York, 1982, pp. 8–16.
- [27] M. FÜRER, *Data structures for distributed counting*, J. Comput. System Sci., 28 (1984), pp. 231–243.
- [28] J. GESKE, D. HUYNH, AND J. SEIFERAS, *A note on almost-everywhere-complex sets and separating deterministic-time-complexity classes*, Inform. Comput., 92 (1991), pp. 97–104.
- [29] O. GOLDBREICH, S. GOLDWASSER, AND S. MICALI, *How to construct random functions*, J. ACM, 33 (1986), pp. 792–807.
- [30] J. HÅSTAD, R. IMPAGLIAZZO, L. A. LEVIN, AND M. LUBY, *A pseudorandom generator from any one-way function*, SIAM J. Comput., 28 (1999), pp. 1364–1396.
- [31] F. HENNIE AND R. STEARNS, *Two-tape simulation of multitape Turing machines*, J. ACM, 13 (1966), pp. 533–546.
- [32] R. IMPAGLIAZZO, V. KABANETS, AND A. WIGDERSON, *In search of an easy witness: Exponential time vs. probabilistic polynomial time*, J. Comput. System Sci., 65 (2002), pp. 672–694.
- [33] R. IMPAGLIAZZO AND A. WIGDERSON, *$P = BPP$ if E requires exponential circuits: Derandomizing the XOR lemma*, in Proceedings of the ACM Symposium on Theory of Computing (STOC), ACM, New York, 1997, pp. 220–229.
- [34] R. IMPAGLIAZZO AND A. WIGDERSON, *Randomness vs. time: De-randomization under a uniform assumption*, J. Comput. System Sci., 63 (2001), pp. 672–688.

- [35] V. KABANETS, *Easiness assumptions and hardness tests: Trading time for zero error*, J. Comput. System Sci., 63 (2001), pp. 236–252.
- [36] V. KABANETS AND J.-Y. CAI, *Circuit minimization problem*, in Proceedings of the ACM Symposium on Theory of Computing (STOC), ACM, New York, 2000, pp. 73–79.
- [37] A. R. KLIVANS AND D. VAN MELKEBEEK, *Graph nonisomorphism has subexponential size proofs unless the polynomial-time hierarchy collapses*, SIAM J. Comput., 31 (2002), pp. 1501–1526.
- [38] K.-I. KO, *On the complexity of learning minimum time-bounded Turing machines*, SIAM J. Comput., 20 (1991), pp. 962–986.
- [39] M. KOUCKÝ, *On Traversal Sequences, Exploration Sequences, and Completeness of Kolmogorov Random Strings*, Ph.D. thesis, Rutgers University, New Brunswick, NJ, 2003.
- [40] M. KUMMER, *On the complexity of random strings*, in Proceedings of the Symposium on Theoretical Aspects of Computer Science (STACS), Lecture Notes in Comput. Sci. 1046, Springer-Verlag, Berlin, 1996, pp. 25–36.
- [41] R. LADNER, *On the structure of polynomial time reducibility*, J. ACM, 22 (1975), pp. 155–171.
- [42] R. E. LADNER AND N. A. LYNCH, *Relativization of questions about log space computability*, Math. Systems Theory, 10 (1976), pp. 19–32.
- [43] L. LEVIN, *Randomness conservation inequalities: Information and independence in mathematical theories*, Inform. and Control, 61 (1984), pp. 15–37.
- [44] M. LI AND P. VITANYI, *Introduction to Kolmogorov Complexity and Its Applications*, Springer-Verlag, New York, 1993.
- [45] L. LONGPRÉ, *Resource Bounded Kolmogorov Complexity, A Link between Computational Complexity and Information Theory*, Ph.D. thesis, Cornell University, Ithaca, New York, 1986.
- [46] D. MARTIN, *Completeness, the recursion theorem and effectively simple sets*, Proc. Amer. Math. Soc., 17 (1966), pp. 838–842.
- [47] D. MICCIANCIO, *Improved cryptographic hash functions with worst-case/average-case connection*, in Proceedings of the ACM Symposium on Theory of Computing (STOC), ACM, New York, 2002, pp. 609–618.
- [48] N. NISAN, *Pseudorandom bits for constant depth circuits*, Combinatorica, 11 (1991), pp. 63–70.
- [49] N. NISAN AND A. WIGDERSON, *Hardness vs. randomness*, J. Comput. System Sci., 49 (1994), pp. 149–167.
- [50] M. RABIN, *Digitalized Signatures and Public-Key Functions as Intractible as Factorization*, MIT Tech. report TR-212, MIT, Cambridge, MA, 1979.
- [51] A. RAZBOROV AND S. RUDICH, *Natural proofs*, J. Comput. System Sci., 55 (1997), pp. 24–35.
- [52] D. RONNEBURGER, *Kolmogorov Complexity and Derandomization*, Ph.D. thesis, Rutgers University, New Brunswick, NJ, 2004.
- [53] S. RUDICH, *Super-bits, demi-bits, and $\mathcal{NP}/\text{qpoly}$ -natural proofs*, in RANDOM, Lecture Notes in Comput. Sci. 1269, Springer-Verlag, Berlin, 1997.
- [54] M. SIPSER, *A complexity theoretic approach to randomness*, in Proceedings of the ACM Symposium on Theory of Computing (STOC), ACM, New York, 1983, pp. 330–335.
- [55] M. SUDAN, L. TREVISAN, AND S. VADHAN, *Pseudorandom generators without the XOR lemma*, J. Comput. System Sci., 62 (2001), pp. 236–266.
- [56] L. TREVISAN, *Construction of extractors using pseudo-random generators*, J. ACM, 48 (2001), pp. 860–879.
- [57] L. TREVISAN AND S. VADHAN, *Pseudorandomness and average-case complexity via uniform reductions*, in Proceedings of the IEEE Conference on Computational Complexity, IEEE Computer Society, Los Alamitos, CA, 2002, pp. 129–138.
- [58] U. VAZIRANI AND V. VAZIRANI, *A natural encoding scheme proved probabilistic polynomial complete*, Theoret. Comput. Sci., 24 (1983), pp. 291–300.
- [59] E. VIOLA, *The complexity of constructing pseudorandom generators from hard functions*, Comput. Complexity, 13 (2004), pp. 147–188.
- [60] H. VOLLMER, *Introduction to Circuit Complexity*, Springer-Verlag, New York, 1999.
- [61] O. WATANABE, *A comparison of polynomial time completeness notions*, Theoret. Comput. Sci., 54 (1987), pp. 249–265.
- [62] O. WATANABE AND S. TANG, *On polynomial-time Turing and many-one completeness in PSPACE*, Theoret. Comput. Sci., 97 (1992), pp. 199–215.
- [63] A. YAO, *Theory and applications of trapdoor functions*, in Proceedings of the IEEE Symposium on Foundations of Computer Science (FOCS), IEEE Computer Society, Los Alamitos, CA, 1982, pp. 80–91.

FULLY DYNAMIC ORTHOGONAL RANGE REPORTING ON RAM*

CHRISTIAN WORM MORTENSEN†

Abstract. We show that there exists a constant $\omega < 1$ such that the fully dynamic d -dimensional orthogonal range reporting problem for any constant $d \geq 2$ can be solved in time $O(\log^{\omega+d-2} n)$ for updates and time $O((\log n / \log \log n)^{d-1} + r)$ for queries. Here n is the number of points stored and r is the number of points reported. The space usage is $O(n \log^{\omega+d-2} n)$. For $d = 2$ our results are optimal in terms of time per operation, and this is the main contribution of this paper. Also for $d = 2$, we give a new improved fully dynamic structure supporting 3-sided queries. The model of computation is a unit cost RAM. We order the coordinates of points using *list order* as defined in the paper.

Key words. range searching, data structures, orthogonal

AMS subject classifications. 68P05, 68P10, 68P15

DOI. 10.1137/S0097539703436722

1. Introduction. Consider a database R of persons where each person has attributes such as an age, a weight, and a height. One of the most fundamental types of queries in the database is to ask for the set of persons in R who have an age between, say, 30 and 50 years, a weight between 80 and 90 kg, and a height between 160 and 170 cm. Data structures supporting these kinds of queries are called orthogonal range reporting structures and have been widely studied during the last 30 years. For surveys see Agarwal and Erickson [1] and Chiang and Tamassia [18]. In this paper we give new upper bounds for the fully dynamic variant of this problem where we in an online setting allow queries to be intermixed with updates which can insert a new person or delete an existing person. In case each person has exactly two attributes our upper bound is optimal in terms of time per operation by a lower bound of Alstrup, Husfeldt, and Rauhe [4].

1.1. Problem definition. In the d -dimensional fully dynamic orthogonal range reporting problem (henceforth the dynamic d -dimensional range reporting problem or just the d -dimensional range reporting problem) we must maintain a set R of at most $O(n)$ d -dimensional points under insertions and deletions. For each point $(x_1, \dots, x_d) \in R$ we must have $x_i \in L_i$ for $1 \leq i \leq d$, where L_i is an ordered set. Given a query $(x'_1, x''_1, \dots, x'_d, x''_d) \in L_1 \times L_1 \times \dots \times L_d \times L_d$ we must report the set $\{(x_1, \dots, x_d) \in R \mid x'_1 \leq x_1 \leq x''_1 \wedge \dots \wedge x'_d \leq x_d \leq x''_d\}$. We assume that if $(x_1, \dots, x_d) \in R$ and $(y_1, \dots, y_d) \in R$ are two different points, then $x_1 \neq y_1 \wedge \dots \wedge y_d \neq y_d$. Our model of computation is a unit cost RAM with word size at least $\log n$ bits. We define the *update time* to be the time it takes to insert a point into or delete a point from R . We consider only solutions to the problem where the time to answer a query has the form $Q + O(r)$, where r is the number of reported points and Q is independent of r , and we say such a solution has *query time* Q . Further, we call the maximum of the update time and the query time the *time per operation*.

*Received by the editors October 28, 2003; accepted for publication (in revised form) October 27, 2005; published electronically April 21, 2006. This paper is based on [31] and [30]. The result of [31] for the orthogonal segment intersection problem is not covered in this paper. Also, we use another method for extending into higher dimensions than in [30].

<http://www.siam.org/journals/sicomp/35-6/43672.html>

†IT University of Copenhagen, Rued Langgaards Vej 7, 2300 Copenhagen S, Denmark (cworm@itu.dk).

We get different problems depending on what we select as L_i . Traditionally, one is only allowed to compare the elements of L_i using a unit cost comparison operation. This is often specified as $L_i = \mathbb{R}$ and makes good sense on a pointer machine. We call this the *comparison order* variant of our problem. Another possibility is to take L_i to be the set of nonnegative integers smaller than m (we assume here $m \leq 2^w$, where w is the word size). We define this as the *m-order* variant. We define the *word order* variant to be the *m-order* variant with $m = 2^w$. All these variants have their shortcomings. In the comparison order variant, there is a trivial lower bound of $\Omega(\log n)$ on the time per operation because this is a lower bound for the dictionary problem on \mathbb{R} . In the *m-order* variant, it is not always possible to create a new attribute between two existing attributes when performing an insertion. For these reasons, we introduce in this paper what we define as the *list order* variant. For any linked list L we order the elements of L such that if $e, e' \in L$, then $e < e'$ if $e \neq e'$ and e appears before e' in L . In the list order variant, we then let L_i be a linked list and require each element of L_i to be the coordinate of exactly one point in R . When a point is inserted into (resp., deleted from) R a new element is then inserted into (resp., deleted from) each L_i for $1 \leq i \leq d$. Using a result by Dietz and Sleator [21] (cited as Theorem 8 in section 5.3) a structure for the comparison order variant can be turned into a structure for the list order variant with the same performance. Conversely, a structure for the list order variant can be turned into a structure for the comparison order variant if we add a term of $O(\log n)$ to the update and query time since such a term allows us to maintain a balanced search tree with the elements of each L_i . Finally, by the use of Theorem 6 cited in section 5.1, a structure for the list order variant can be turned into a structure for the word order variant if we add a term of $O(\sqrt{\log n / \log \log n})$ to the update and query time.

1.2. Our results. We show (Theorem 32) that there exists a constant $\omega < 1$ such that for any constant $d \geq 2$ the d -dimensional range reporting problem in the list order variant can be solved with update time $O(\log^{\omega+d-2} n)$, query time $O((\log n / \log \log n)^{d-1})$, and space $O(n \log^{\omega+d-2} n)$. The time bounds are worst case and the solution is deterministic. In the comparison order variant, the discussions of section 1.1 imply that the same result holds for $d \geq 3$ and that for $d = 2$ the time per operation becomes $\Theta(\log n)$.

For $d = 2$ we also consider queries of the restricted form $(x'_1, x''_1, x'_2, x''_2) \in L_1 \times L_1 \times L_2 \times L_2$, where x'_2 is the minimal element of L_2 . Such queries are called *3-sided*, whereas queries where x'_2 can take any value in L_2 are called *4-sided*. We show (Theorem 24) that the 2-dimensional range reporting problem in the list order variant with 3-sided queries only can be solved with update time $O(\log^\omega n)$, query time $O(\log n / \log \log n)$, and space $O(n)$. Here also, the time bounds are worst case and the solution is deterministic.

The data structures of this paper are too complicated and the constants involved too large to be useful in practice. However, some of the ideas of the paper may still have practical implications (see section 9 on open problems).

1.3. Relation to other work. The static 2-dimensional range reporting problem in the comparison order variant can be solved with query time $O(\log^2 n)$ and space $O(n \log n)$ using *range trees*. The range tree data structure was independently discovered by Bentley, Lee, Lueker, Shamos, Willard, and Wong (see [12, 9, 10, 25, 26, 40, 42]). It is commonly recognized that Bentley and Shamos were chronologically the first authors to publish their results [12, 9]. In a dynamic setting, their data structure was known to have an $O(\log^3 n)$ query time and an $O(\log^2 n)$ update time

using some version of the Logarithmic update method—sometimes called the static-to-dynamic transformation [11, 34, 36]. It was observed by Lueker and Willard (first independently and then in a joint paper [26, 40, 46]) that the query time of this dynamic method for range trees could be reduced to $O(\log^2 n)$ time (with no sacrifice in update time) using the $BB(\alpha)$ dynamic method. Willard [42] observed how to extend range trees with “downpointers” so that the query time could be further reduced to $O(\log n)$ time in the static case. The downpointer discovery by Willard [42] should not be confused with the $BB(\alpha)$ dynamic method (that was jointly coauthored by Lueker and Willard (see [26, 40, 46])). Because the two discoveries of the $BB(\alpha)$ dynamic method and the downpointer were made nearly simultaneously, several articles reviewing the prior literature have often inadvertently confused them.

Chazelle and Guibas [17] generalized downpointers as well as other ideas into a data structuring technique called fractional cascading. Mehlhorn and Näher [29] made fractional cascading dynamic with amortized time bounds on updates. They used this to develop a data structure for the 2-dimensional range reporting problem in the comparison order variant with update and query time $O(\log n \log \log n)$ on a pointer machine. Dietz and Raman [20] removed amortization from the results of [29] on a RAM. Using range trees the structure from [29] can be extended to $d > 2$ dimensions giving update and query time $O(\log^{d-1} n \log \log n)$ and space usage $O(n \log^{d-1} n)$. Until now, this was the fastest known solution in terms of time per operation (even in the n -order variant). Range trees with slack parameter [28, 37] can be used to get a structure with polylogarithmic update and query time and with space usage $O(n(\log n / \log \log n)^{d-1})$. Until now, no solution with space usage $o(n(\log n / \log \log n)^{d-1})$ and polylogarithmic time per operation was known (see end of paragraph for very recent work). For $d = 2$ a solution with linear space usage $O(n)$, update time $O(\log n)$, and query time $O(n^\epsilon)$ for a constant $\epsilon > 0$ is known [24]. Also, for $d = 2$, if only insertions or deletions (but not both) are supported Imai and Asano [23] described a solution using space $O(n \log n)$ and time $\Theta(\log n)$ per operation. Very recently, Nekrich [33] dynamized a structure by Alstrup, Brodal, and Rauhe [3] and obtained a structure for the d -dimensional dynamic range reporting problem with update time $O(\log^d n)$, query time $O(\log^{d-1} n)$, and space usage $O(n \log^{d-2+\epsilon})$ for any constant $\epsilon > 0$.

Let U be the update and Q be the query time of an orthogonal range reporting structure. For word size polylogarithmic in n Alstrup, Husfeldt, and Rauhe [4] showed that $Q = \Omega(\log n / \log(U \log n))$. This lower bound holds in the cell probe model of computation (which is stronger than the RAM model) for the amortized cost per operation, for 3-sided queries, and for the n -order variant. It follows that with polylogarithmic update time, the query time becomes $\Omega(\log n / \log \log n)$. Together with our new upper bound, this lower bound gives that the time per operation for the 4-sided 2-dimensional range reporting problem in the list order variant and the m -order variant is $\Theta(\log n / \log \log n)$ for $m \geq n$. As already mentioned our results also imply a bound of $\Theta(\log n)$ in the comparison order variant. The time per operation for the 2-dimensional range reporting problem in all the order variants of section 1.1 is thus now completely understood, and this is the major contribution of this paper.

McCreight [27] developed the priority search tree which solves the 2-dimensional range reporting problem in the comparison order variant with 3-sided queries only on a pointer machine. The update and query time of the priority search tree is $O(\log n)$ and it uses space $O(n)$. Willard [45] modified the priority search tree and obtained a structure using time $O(\log n / \log \log n)$ for updates and queries in the word order variant on a RAM. In this paper, we reduce the update time for this problem further

without increasing the query time or the space usage. Also, we generalize the result to the list order variant. As did Willard, we use a modified version of the priority search tree.

A longstanding open problem mentioned by McCreight [27] is whether it is harder to make a fully dynamic structure supporting 4-sided than it is for 3-sided queries. This paper together with the lower bound of [4] partly answers this question by showing that on the RAM the time per operation needed for the two problems is the same in all the order variants described in section 1.1. Whether the space usage for structures supporting 4-sided queries can be reduced to linear as for 3-sided queries is still open. This is even the case in the static case where the best known structures with polylogarithmic query time and constant time for each reported point use space $O(n \log^\epsilon n)$. The query time is $O(\log n)$ for the comparison order variant [15] and $O(\log \log n)$ for the n -order variant [3].

On the pointer machine in the comparison order variant, Chazelle [16] has shown that any static or dynamic structure for the 2-dimensional range reporting problem with polylogarithmic query time must use space $\Omega(n(\log n / \log \log n)^{d-1})$. For $d = 2$ this bound is matched by a static structure with query time $O(\log n)$ and space usage $O(n \log n / \log \log n)$ [14]. For $d > 2$ the best known upper bound with matching space usage has query time $O(\log^{d-1+\epsilon} n)$ for any constant $\epsilon > 0$ also in the static case [16].

In the input/output (I/O) model of computation with block size B and in 2 dimensions, a structure for 3-sided queries using space $O(n/B)$ disk blocks supporting updates in time $O(\log_B n)$ and queries in time $O(\log_B n + r/B)$ when r points are reported is described by Arge, Samoladas, and Vitter [6]. This is optimal in terms of time per operation in the comparison order variant. For 4-sided queries they describe a solution supporting updates in time $O((\log_B n) \log(n/B) / \log \log_B n)$ and queries in time $O(\log_B n + r/B)$ when r points are reported. The structure uses space $O((n/B) \log(n/B) / \log \log_B n)$ disk blocks.

1.4. Outline of paper. After this introduction we continue with preliminaries in section 2. In section 3 we give various definitions related to 2-dimensional range reporting. Also, we give an outline of how the results of this paper are obtained. In section 4 we give probably the most central construction of this paper, and this construction may be of independent interest. In section 5 we review various well-known data structures which we use in this paper. Also, we develop some new results on top of these structures. In section 6 we develop our solution to the 2-dimensional range reporting problem with 3-sided queries only. In section 7 we develop our solution to the 4-sided 2-dimensional range reporting problem. In section 8 we extend the solution of section 7 to higher dimensions. The paper concludes with open problems in section 9 and acknowledgments.

2. Preliminaries. For the rest of this paper we define $[i \dots j] = \{k \in \mathbb{Z} | i \leq k \leq j\}$ and we let \log denote the base-two logarithm. We will measure space in bits. We will use $\log^i \log n$ as an abbreviation for $(\log \log n)^i$. All time bounds are worst case and all structures are deterministic if not otherwise noted.

A linked list is a list where the elements are doubly linked with pointers. We make no assumptions about how the elements are laid out in memory. If the list supports insertions, we can insert a new element in it if we have a pointer to the existing element the new element should be inserted before or after. If the list supports deletions, we can delete an element if we have a pointer to it. We assume the elements of a linked list and the children of a node in a tree are ordered from left to right.

We will not distinguish between a data structure and the set of elements it contains. As a consequence, $|G|$ is the number of elements in the data structure G and $e \in G$ means that e is an element in G .

A function f is said to be *subadditive* if for integers $m, n \geq 0$ we have $f(m) + f(n) \leq f(m + n) + O(1)$. Suppose for each $n \geq 0$ we have a data structure G_n which uses space $f(n)$ bits and which can contain at most n elements. It follows (by induction on k) that if f is subadditive, then storing the elements of G_n in k different structures $G_{n(1)}, \dots, G_{n(k)}$ uses space at most $f(n) + O(k)$ bits, where $n = \sum_{i=1}^k n(i)$.

We will assume that the number of points which can be contained in the range reporting structures we ultimately design is bounded by $O(N)$. We note that N was called n in the introduction and that we have assumed a word size of at least $\log N$ bits. All the data structures we create in this paper are a part of one of our ultimate structures and can contain at most $O(N)$ elements. When constructing such a data structure, we often state that we can compute some function f from $[0 \dots O(N)]$ to $[0 \dots O(N)]$ in constant time using a *global lookup table*. What we mean by this is the following. As a part of the ultimate structure the data structure is a part of, we keep a table of size at most $O(N \log N)$ bits which is constructible in time $O(N)$. The function f can then be computed by performing a constant number of lookups in this table. We are allowed to assume the existence of such tables by using the global rebuilding of Overmars [34]. For brevity, we will omit the details in constructing and looking up in these tables.

3. Definitions and outline of constructions. The constructions of this paper involve many different kinds of range reporting structures. In section 3.1 we will define notation which captures almost all the kinds of 2-dimensional structures we look at. We will use this notation throughout the rest of the paper. Next, in section 3.2 we will give an outline of the constructions that lead to the results of section 1.2.

3.1. Definitions. We now define what it means for a range reporting structure R to have type $R^\tau(t_x : n_x, t_y : n_y)$, where n_x and n_y are integers, $\tau \in \{3, 4\}$, and $t_x, t_y \in \{d, s, s'\}$. R must have an *x-axis* $R.L_x$ and a *y-axis* $R.L_y$. The rest of this section is parametrized over a parameter $z \in \{x, y\}$. Each sentence of this section that contains a z should then be read twice: first with $z = x$ and then with $z = y$. So we can state which axes R must have as follows. R must have a *z-axis* $R.L_z$. R contains elements $e \in R$ which are also called points. A point $e \in R$ has a z -coordinate $e.z \in R.L_z$.

If $t_z = d$, we say the z -axis is *dynamic* and in this case, the z -axis is a linked list with $O(n_z)$ elements. Also, when $t_z = d$ we require that each element of $R.L_z$ is the z -coordinate of exactly one point. If $\tau = 4$, the structure supports 4-sided queries and if $\tau = 3$, the structure support 3-sided queries as defined in section 1.2. We will make this more precise in a moment. As an example, a structure with type $R^4(d : n, d : n)$ is a 4-sided 2 dimensional range reporting structure in the list order variant, and the structure can contain $O(n)$ points.

If $t_z = s$ or $t_z = s'$, we say the z -axis is *static* and in this case, $R.L_z$ is the set $[0 \dots \Theta(n_z)]$ (which is not required to be explicitly stored). If $t_z = s$, there can be at most one point in R with a given z -coordinate; if $t_z = s'$, we have no such restriction. We require that no two points share the same x - and y -coordinate. More precisely, we require that $e, e' \in M$ and $e.x = e'.x$ and $e.y = e'.y$ imply $e = e'$. As an example, a structure with type $R^4(s : n, s : n)$ can contain at most $O(n)$ points, and no two points can share an x - or y -coordinate. On the other hand, a structure with type

$R^4(s':n, s':n)$ can contain $O(n^2)$ points.

We support updates in the form of inserting a new point into R and deleting an existing point from R . From our requirements above it follows that if $t_z = d$, we will then also have to insert an element into and delete an element from $R.L_z$, respectively. For $x_1, x_2 \in R.L_x$ and $y_1, y_2 \in R.L_y$ we support a query (x_1, x_2, y_1, y_2) which must report the set $\{e \in R \mid x_1 \leq e.x \leq x_2 \wedge y_1 \leq e.y \leq y_2\}$. We say that the points in this set are in the *query region*. If $\tau = 3$, we require that y_1 in such a query can be assumed to be the minimal element of $R.L_y$. An implication of our definitions is that for $\tau = 4$ structures of type $R^\tau(s:n, d:n)$ and $R^\tau(d:n, s:n)$ are identical modulo renaming of axes, while this is not the case for $\tau = 3$.

Suppose R can contain at most n points for n small compared to N . In case R has static axes, we will sometimes allow updates to be performed in subconstant time per update by packing several updates into a single word. In this case, the user must assign to each point $e \in R$ a unique identifier or just *id* $e.id$ of size $O(\log n)$ bits. For each point e inserted into R the user must then in the words describing the update provide the triple $(e.x, e.y, e.id)$. For each point $e \in R$ deleted from R , the user must, again in the words describing the update, provide the tuple $(e.x, e.y)$. Finally, in connection with queries the id of the reported points are given to the user (using constant time per point reported).

We say R has *performance* (U, Q, S, t) if R supports $O(t)$ updates in time $O(U)$, has query time $O(Q)$, and has uses space $O(S)$ bits. S is allowed only to depend on the type of R and not on the number of points stored in R . It follows that if R can contain n points, we must have $S = \Omega(n \log n)$. We write (U, Q, S) as an abbreviation for $(U, Q, S, 1)$.

3.2. Outline of constructions. We now have the terminology to give an outline of the constructions leading to the results of section 1.2. Our construction consists of a number of steps which we will refer to during the paper. First, we give an outline of how we prove Theorem 24. This theorem gives our structure with type $R^3(d:N, d:N)$ proving the second result in section 1.2. All the steps O1 to O4 are performed in section 6.

- O1 First, we create a range reporting structure for the case we have only few points. For sufficiently small u compared to N (we have $u = 2^{\log^\epsilon N}$ for a constant $\epsilon > 0$) the structure has type $R^3(s:u, s:u)$, has subconstant update time, and small query time. The result of this step is in Lemma 20.
- O2 We then give the structure of step O1 a dynamic y-axis transforming it into a structure with type $R^3(s:u, d:u)$. What we lose is the subconstant update time; this structure has only constant update time. The query time is unchanged. The technique used is to assign a label of size $O(u)$ to each element on the y-axis and then to relabel elements on insertions. The result of this step is in Lemma 22.
- O3 By creating a fixed-shape tree (actually, a variant of a priority search tree) with a structure of step O2 in each internal node, we get a structure with type $R^3(s:n, d:n)$ for any $n \geq u$. The result of this step is in Lemma 23.
- O4 We then create our final structure with type $R^3(d:n, d:n)$. This is done by grouping points on the x-axis into blocks with a polylogarithmic number of points in each block. The blocks are then numbered respecting the order of the blocks, and then they are inserted into the structure of step O3 using these numbers. The result of this step is in Theorem 24.

Next, we give an outline of how we prove Theorem 32. This theorem gives a structure

with type $R^4(d : N, d : N)$ and a similar structure for higher dimensions. This is the first result from section 1.2. Steps O5 to O10 are performed in section 7, while step O11 is performed in section 8.

- O5 We extend the construction in step O1 to give a structure with type $R^4(s' : u, s : u)$. As in step O1 this structure has subconstant update time and small query time. The result of this step is in Lemma 25.
- O6 We use the same construction as in step O2 to get a dynamic x-axis. More precisely, we get a structure with type $R^4(d : u, s' : u)$. The result of this step is in Lemma 26.
- O7 By maintaining a predecessor structure, we transform this into a structure with a static but long x-axis. The structure we get has type $R^4(s' : n, s : u)$ for any $n, u \leq n \leq N$. The result of this step is in Lemma 27.
- O8 We then use the construction of section 4 to give the structure of step O7 type $R^4(s' : n, s' : u)$ almost without reducing its query time or increasing its update time. The result of this step is in Lemma 28.
- O9 We then give this structure a dynamic x-axis creating a structure R with type $R^4(d : n, s' : u / \log^3 n)$. This is done by grouping points into blocks of size $O(u)$ on the x-axis. In each block, we keep a structure of step O6. As in step O4 we maintain a numbering of the blocks and we insert each block into the structure of step O8 using the numbers as x-coordinate. The result of this step is in Lemma 29.
- O10 We then construct our structure with type $R^4(d : n, d : n)$. This is done by constructing a variant of a range tree using the structure of step O9 in some nodes and the structure of step O4 in some nodes of the tree. The result of this step is in Theorem 31.
- O11 Finally, we extend the structure of step O10 to higher dimensions. The result of this step is in Theorem 32.

4. From few to many points. In this section, we give the construction used in step O8 in section 3.2. As mentioned earlier, this is probably the most central construction of this paper. The construction allows us to create a 4-sided range reporting structure which can contain many points from one that can contain few points. The construction has applications besides the one in this paper [32, 31], and for this reason we will give a slightly more general form of the construction than we need here.

We now define what it means for a data structure G to have type (n, Y, \mathcal{Y}) , where n is an integer and \mathcal{Y} is a set of subsets of Y . G contains elements $e \in G$, where $e.x$, $0 \leq e.x < n$, is the *position* of e and $e.y \in Y$ is the *height* of e . Two different elements in G are not allowed to have both the same position and height implying $|G| \leq n|Y|$. We are allowed to update G by inserting a new element or by deleting an existing element. Further, for $0 \leq i, j < n$ and $q \in \mathcal{Y}$ we can ask the query $\text{report}(G, i, j, q)$ which must report the set $\{e \in G \mid i \leq e.x \leq j \wedge e.y \in q\}$. As in the rest of the paper, the query time is said to be $O(Q)$ if reporting this set takes time $O(Q)$ plus constant time for each reported point. We will then show the following theorem.

THEOREM 1. *Suppose we have a data structure G' with type (n, Y, \mathcal{Y}) and the restriction*

$$(1) \quad e, e' \in G' \wedge e.y = e'.y \implies e = e'.$$

Suppose further G' uses space $O(S)$ bits, has query time $O(Q)$, and update time $O(U)$. Then we can make a structure G with type (n, Y, \mathcal{Y}) without the restriction (1) which

uses space $O(Sn \log \log n)$ bits, has query time $O(Q \log \log n)$, and has update time $O(U \log \log n)$.

Observe that since G' can contain $|Y|$ elements we must have $S = \Omega(|Y| \log n)$. In this paper, we use only the following corollary to the theorem.

LEMMA 2. *Suppose there exists a structure with type $R^4(s' : n, s : u)$ and performance (U, Q, S) . Then there exists a structure with type $R^4(s' : n, s' : u)$ and performance $(U \log \log n, Q \log \log n, Sn \log \log n)$.*

Proof. The lemma is just another way of phrasing Theorem 1 if we select $Y = [0 \dots O(u)]$ and $\mathcal{Y} = \{[y_1 \dots y_2] \mid 0 \leq y_1, y_2 \leq O(u)\}$. \square

The rest of this section is devoted to the proof of Theorem 1. We need the following theorem which is shown by van Emde Boas et al. (see [39, 38]).

THEOREM 3. *Let w be the word size in bits. There exists a data structure called a VEB which can maintain a collection of $n \leq U \leq 2^w$ elements having keys in $[0 \dots U]$ which uses space $O(U \log U)$ bits and supports updates as well as predecessor and successor queries in time $O(\log \log U)$.*

We create a VEB of Theorem 3 for each element $y \in Y$. Each element $e \in G$ is inserted at position $e.x$ into the VEB for height $e.y$. Using these VEBs we link all elements with the same height together in order according to their positions. This will use space $O(n|Y| \log n) = O(Sn)$ and insertions and deletions can be performed in time $O(\log \log n)$.

We maintain a data structure S_n which we develop in the following. The data structure S_n contains triples $(e, x, y) \in S_n$, where e is an arbitrary pointer, x is an integer in $[0 \dots n - 1]$, and $y \in Y$ is the height of the triple. We maintain S_n such that $(e, e.x, e.y) \in S_n$ iff $e \in G$. We support a special `reportsub`(S_n, i, j, q) query which has the following properties:

1. `reportsub`(S_n, i, j, q) \subseteq `report`(G, i, j, q).¹
2. If for given y there exists an element $e \in$ `report`(G, i, j, q) with $i \leq e.x \leq j$ and $e.y = y$, then `reportsub`(S_n, i, j, q) contains at least one such e .
3. S_n has query time $O(Q)$.

A `report`(G, i, j, q) query can then be answered as follows. First, we perform a `reportsub`(S_n, i, j, q) query. Properties 1 and 3 ensure that we will not use too much time on this. For each height $y \in Y$ for which there exists an element $e \in$ `reportsub`(S_n, i, j, q) with height y we follow the pointers from e maintained by the VEB for height y to report the rest of the elements with this height. Property 2 ensures that this will report all elements.

We now describe the structure S_n . To avoid tedious details, we assume n has the form $n = 2^{2^m}$ for an integer $m \geq 0$. We observe that if $n > 2$ has this form, then \sqrt{n} has same form. The structure S_n is somewhat similar to a VEB, and we define it inductively on n . If $S_n = \emptyset$, the recursion stops. Otherwise, we keep an array $S_n.\text{min}$ (resp., $S_n.\text{max}$) indexed by Y . For each $y \in Y$ we store the triple $(e, x, y) \in S_n$ with minimal (resp., maximal) value of x in $S_n.\text{min}[y]$ (resp., $S_n.\text{max}[y]$) if any. We also keep an array $S_n.\text{bottom}$ indexed by $[0 \dots \sqrt{n} - 1]$ where in each entry we store a recursive structure with type $S_{\sqrt{n}}$. We store each triple $(e, i, y) \in S_n \setminus (S_n.\text{min} \cup S_n.\text{max})$ as $(e, i \bmod \sqrt{n}, y)$ in $S_n.\text{bottom}[i \text{ div } \sqrt{n}]$.² Finally, we keep a single recursive structure $S_n.\text{top}$ also with type $S_{\sqrt{n}}$. If for $y \in Y$ the structure $S_n.\text{bottom}[i]$ contains exactly one triple $(e, x, y) \in S_n$ with height y , we store (e, i, y) in $S_n.\text{top}$. We note that in

¹In section 4 we will not distinguish between a query and its answer.

²For integers i and d we define $i \bmod d$ and $i \text{ div } d$ to be the unique integers such that $0 \leq i \bmod d < d$ and $i = d(i \text{ div } d) + i \bmod d$.

this case e is stored in both $S_n.\text{bottom}$ and in $S_n.\text{top}$. If $S_n.\text{bottom}[i]$ contains more than one triple with height y , we store (e', i, y) in $S_n.\text{top}$, where e' is a pointer to the recursive structure in $S_n.\text{bottom}[i]$.

Inserting a triple (e, x, y) into S_n : If S_n contains at most one triple with height y , we just update $S_n.\text{min}[y]$ and $S_n.\text{max}[y]$ and we are done. Otherwise, we first check if (e, x, y) should go into $S_n.\text{min}[y]$ (resp., $S_n.\text{max}[y]$), and if this is the case we interchange (e, x, y) and the triple in $S_n.\text{min}[y]$ (resp., $S_n.\text{max}[y]$). Let T be the structure in $S_n.\text{bottom}[x \text{ div } \sqrt{n}]$. We then insert $(e, x \bmod \sqrt{n}, y)$ into T . Let m be the number of triples in T with height y after this insertion. If $m = 1$, we insert $(e, x \text{ div } \sqrt{n}, y)$ into $S_n.\text{top}$. If $m = 2$, there is a triple $(e', x \text{ div } \sqrt{n}, y)$ in $S_n.\text{top}$, and we replace (see next paragraph) this triple with the triple $(e'', x \text{ div } \sqrt{n}, y)$, where e'' is a pointer to T . We observe that when we update $S_n.\text{top}$ then T has at most two triples with height y and the update in T is performed by just accessing $T.\text{min}$ and $T.\text{max}$. It follows that we perform only a nonconstant number of updates in at most one recursive structure.

Replacing a triple $(e, x, y) \in S_n$ with the triple (e', x, y) : If $(e, x, y) = S_n.\text{min}[y]$ (resp., $(e, x, y) = S_n.\text{max}[y]$), we just update $S_n.\text{min}[y]$ (resp., $S_n.\text{max}[y]$) and we are done. Otherwise, let T be the structure in $S_n.\text{bottom}[x \text{ div } \sqrt{n}]$. First, we replace in T $(e, x \bmod \sqrt{n}, y)$ with $(e', x \bmod \sqrt{n}, y)$. Next, if T has exactly one triple with height y , we replace $(e, x \text{ div } \sqrt{n}, y)$ with $(e', x \text{ div } \sqrt{n}, y)$ in $S_n.\text{top}$. As with insertions we note that we need only to perform a nonconstant number of updates in at most one recursive structure.

Deleting a triple (e, x, y) from S_n : If S_n has at most two triples with height y , we just update $S_n.\text{min}[y]$ and $S_n.\text{max}[y]$ and we are done. Suppose therefore that S_n contains at least three triples with height y . We split into three cases: Case 1: In this case $(e, x, y) = S_n.\text{min}[y]$. Let $(e', t, y) = S_n.\text{top}.\text{min}[y]$, $(e'', l, y) = S_n.\text{bottom}[t].\text{min}[y]$ and $i = l + t\sqrt{n}$. Then (e'', i, y) is the triple in $S_n \setminus (S_n.\text{min} \cup S_n.\text{max})$ with height y which has the minimal value of i . Instead of deleting (e, x, y) from S_n we delete (e'', i, y) and afterwards we set $S_n.\text{min}[y]$ to (e'', i, y) . Case 2: In this case $(e, x, y) = S_n.\text{max}[y]$ and this case is handled in a symmetric way to Case 1. Case 3: In this case (e, x, y) is not equal to $S_n.\text{min}[y]$ or $S_n.\text{max}[y]$. The case is handled in a way similar to insertions: Let T be the structure in $S_n.\text{bottom}[x \text{ div } \sqrt{n}]$. We then delete $(e, x \bmod \sqrt{n}, y)$ from T . Let m be the number of triples in T with height y after this deletion. Suppose $m = 1$ and let (e', i, y) be the triple with height y in T . Then there is a triple $(e'', x \text{ div } \sqrt{n}, y)$ in $S_n.\text{top}$, where e'' is a pointer to T , and we replace this triple with the triple $(e', x \text{ div } \sqrt{n}, y)$. If $m = 0$, we delete $(e, x \text{ div } \sqrt{n}, y)$ from $S_n.\text{top}$. Again we observe that we need only to perform a nonconstant number of updates in at most one recursive structure.

What remains is to describe how to answer a `reportsub` query. In order to do this, we in addition to $S_n.\text{min}$ (resp., $S_n.\text{max}$) maintain a structure $S_n.\text{min}'$ (resp., $S_n.\text{max}'$) with the type of the structure given to Theorem 1. We maintain $S_n.\text{min}'$ (resp., $S_n.\text{max}'$) such that $S_n.\text{min}'$ (resp., $S_n.\text{max}'$) contains an element e with position x , height y , and a value $e.v = e'$ iff $S_n.\text{min}$ (resp., $S_n.\text{max}$) contains the triple (e', x, y) .

Answering a `reportsub`(S_n, i, j, q) query: If $i > j$ or $S_n = \emptyset$, we report nothing. Otherwise, let M be the set of pairs $\{(e.v, e.y) \mid e \in \text{report}(S_n.\text{min}', i, j, q) \cup \text{report}(S_n.\text{max}', i, j, q)\}$. We then iterate the following as long as possible: Take a pair $(e, y) \in M$, where e points to a recursive structure T in our induction on n , and replace the pair with (e', y) and (e'', y) assuming $T.\text{min}[y] = (e', x', y)$ and $T.\text{max}[y] = (e'', x'', y)$. We observe that the time we spend on performing these replacements is proportional to $|M|$. After this, report the elements $\{e \mid \exists y.(e, y) \in M\}$.

If $i = 0$ or $j = n - 1$, we stop. Otherwise, if $i \operatorname{div} \sqrt{n} = j \operatorname{div} \sqrt{n}$, we perform a `reportsub`($S_n.\text{bottom}[i \operatorname{div} \sqrt{n}], i \bmod \sqrt{n}, j \bmod \sqrt{n}, q$) query. Finally if $i \operatorname{div} \sqrt{n} \neq j \operatorname{div} \sqrt{n}$, we split the query into the three queries `reportsub`($S_n.\text{bottom}[i \operatorname{div} \sqrt{n}], i \bmod \sqrt{n}, \sqrt{n} - 1, q$), `reportsub`($S_n.\text{top}, i \operatorname{div} \sqrt{n} + 1, j \operatorname{div} \sqrt{n} - 1, q$), and `reportsub`($S_n.\text{bottom}[j \operatorname{div} \sqrt{n}], 0, j \bmod \sqrt{n}, q$). We note that the first and the last of these three queries are answered by just looking at the `min`, `min'`, `min`, and `max'` fields in the recursive structure. To analyze the space usage of S_n we need the following lemma.

LEMMA 4. *Suppose p is defined by $p(0) = 2$ and $p(h) = 2 + (1 + 2^{2^{h-1}})p(h - 1)$ for $h \geq 1$. Then $p(h) \leq (h + 1)2^{2^h}$.*

Proof. We show the lemma by induction on h . For $h = 0$ the lemma is trivially true. Suppose $h \geq 1$ and that the lemma is true for less h . Then we have $p(h) \leq 2 + (1 + 2^{2^{h-1}})h2^{2^{h-1}} = 2 + h2^{2^{h-1}} + h2^{2^h} \leq 2^{2^h} + h2^{2^h} = (h + 1)2^{2^h}$.

With p as in the lemma we observe that $p(\log \log n)$ is the number of `min`, `max`, `min'`, and `max'` structures we need and the lemma then bounds the space usage of S_n to $O(S_n \log \log n)$ bits as desired.

The running time of the update operations in S_n is $O(U \log \log n)$. This follows from the observation that when an update operation operates in more than one recursive structure, it does a nonconstant number of updates in at most one of these. A similar argument shows that the S_n has query time $O(Q \log \log n)$ concluding our proof of Theorem 1. \square

5. Lists, trees, and predecessor. In sections 5.1 to 5.5 we review various well-known data structures for lists, trees, and the predecessor problem. In sections 5.6 and 5.7 we develop some new results based on some of these well-known structures.

5.1. The predecessor problem. In this section we review various data structures for the predecessor problem. We have already cited one such structure, namely the VEB of Theorem 3. Combining the hashing scheme of [22] with Willard [41] we get the following structure.

LEMMA 5. *If we allow randomization, the space usage of Theorem 3 can be reduced to $O(n \log U)$ bits. The update time is then with probability $1 - n^{-c}$ for any desired constant $c > 0$.*

Building on Beame and Fich [8], Andersson and Thorup [5] have shown the following theorem.

THEOREM 6. *There exists a data structure called a BFAT which can maintain a collection of n elements having keys in $[0 \dots U]$. The structure uses space $O(n \log U)$ bits and supports updates as well as predecessor and successor queries in time $O(\min(\sqrt{\log n / \log \log n}, \log \log n \log \log U / \log \log \log U))$. U is called the universe size and we require $U = 2^{O(w)}$, where w is the word size in bits.*

We will take the time usage of Theorem 6 to be $O(\log^2 \log U)$ (recall that we write $\log^i \log U$ for $(\log \log U)^i$). This is possible since we will always have $n \leq U$ and thus $\log \log n \log \log U / \log \log \log U \leq \log^2 \log U$.

5.2. WBB trees. In this section we review WBB trees (weight balanced B-trees). Variants of these trees have been described by Willard [43], Dietz [19], and Arge and Vitter [7]. The description here is similar to the one in [7], where more details can be found.

Like an (a, b) -tree a WBB tree is a multibranching search tree where (1) elements are kept in the leaves, (2) keys to guide the search are kept in internal nodes, and (3) all leaves have the same depth (and height 0). We support updates to the tree in the form of insertions (not deletions) of elements. We define the weight of an internal

node as the number of elements in the leaves descendant to the node. The tree is parametrized over a *branching parameter* $d \geq 8$ and a *leaf parameter* $k \geq 1$. A leaf node must contain between k and $2k - 1$ elements. When an insertion makes a leaf node contain $2k$ elements it is split into two leaves, each with k elements. An internal node v at height h must have weight between kd^h and $4kd^h$, except if v is the root, in which case we require it only to have weight at most $4kd^h$. When an insertion gives v weight exactly $3kd^h$ a key in v is *marked* such that there is at least kd^h elements descendant to v both to the left and to the right of the marked key. Such a key exists because $(k3d^h - k4d^{h-1})/2 = kd^h(3 - 4/d)/2 \geq kd^h$. When v gets weight exactly $4kd^h$ it is split at the marked key. When a node w (leaf or internal) splits, a new child and a new key are inserted into the parent of w . If w is the root node, a parent of w is first created as new root. The following lemma gives central properties for WBB trees.

LEMMA 7. *Let T be a WBB tree with branching parameter d , leaf parameter k , and n elements. Then we have the following:*

1. *A leaf contains between k and $2k - 1$ elements.*
2. *Internal nodes have degree $O(d)$.*
3. *T has height at most $\min(\log n, O(\log n / \log d))$.*

Further, let $v \in T$ be an internal node at height $h \geq 1$. Then we have the following:

4. *At most one key in v is marked and when v splits it is at the marked key.*
5. *There are at most $4kd^h$ and at least kd^{h-1} elements descendant to v .*
6. *Exactly kd^h insertions are performed in leaves descendant to v from the time when a key in v is marked until the time when v splits.*

As observed in [7], in WBB trees we can maintain secondary structures in each internal node v containing all elements in leaves descendant to v and at the same time have worst case time performance on updates: When a key in v is marked, we can start to build the secondary structures for the two nodes v will eventually split into. Each time an update is performed in a leaf descendant to v we move a constant number of elements to the two new secondary structures. Items 5 and 6 in Lemma 7 ensure that we have sufficient time to do this and item 4 of the lemma ensures that we know which elements to put into which of the two new secondary structures. We finally remark that we can assume there are pointers between an internal node and its children.

5.3. The list order problem. The following theorem is shown by Dietz and Sleator [21]. This theorem was mentioned in the introduction as the theorem allowing us to transform a structure for the list order variant into a structure for the comparison order variant.

THEOREM 8. *There exists a linear sized data structure for a linked list supporting deletion and insertion of list elements in constant time such that we for given list elements e and e' in constant time can determine if $e < e'$.*

5.4. The online list labeling problem. We define the online list labeling problem as follows (other similar definitions are used in other papers; see, e.g., [47]). Let L be a linked list with at most n elements supporting insertions and deletions of elements. We must for each element $e \in L$ maintain an integer label $e.\text{label}$ of size $O(n)$ such that for $e, e' \in L$,

$$(2) \quad e < e' \implies e.\text{label} < e'.\text{label}.$$

An algorithm solving this problem is allowed to change the label of (relabel) elements when an element is inserted into or deleted from L . We require (2) to be maintained

during this relabeling. If the algorithm relabels at most m elements on each insertion into or deletion from L , we say that the algorithm has *relabeling cost* m . The following theorem is shown by Willard [44].

THEOREM 9. *There exists an algorithm for the online list labeling problem with relabeling cost $O(\log^2 n)$ and space usage $O(n \log n)$ bits, which uses time $O(\log^2 n)$ for insertions and deletions.*

5.5. List block balancing. The following theorem is proved by Dietz and Sleator [21, Theorem 5]. Recall that the Harmonic numbers $H_n = \sum_{i=1}^n 1/i$ satisfy $H_n = \Theta(\log n)$.

THEOREM 10. *Suppose x_1, \dots, x_n are variables which are initially zero. Iterate the following two steps. (1) Add a nonnegative value to each x_i such that $\sum_{i=1}^n x_i$ increases by at most one. (2) Set a largest x_i to zero. Then for all i , $1 \leq i \leq n$, we always have $x_i \leq 1 + H_{n-1}$.*

The next lemma follows directly from this theorem.

LEMMA 11. *Suppose $k \geq 1$ and that \mathcal{C} is a collection of at most n sets and that initially $\mathcal{C} = \{\emptyset\}$. Iterate the following two steps. (1) Add at most k elements to the sets of \mathcal{C} . (2) If $M \in \mathcal{C}$ is a largest set in \mathcal{C} and $|M| \geq 5k(1 + H_{n-1})$, then split M into two sets each of size at least $2k(1 + H_{n-1})$ and let these replace M in \mathcal{C} . Then for all $M \in \mathcal{C}$ we always have $|M| \leq 6k(1 + H_{n-1})$.*

We now describe a general technique used in [21] which we will call *list block balancing* with *block size* $s \geq \log^3 n$. We use this technique as a part of the steps O4 and O9 in section 3.2 where we group elements of an axis into numbered blocks and use this to convert a static axis to a dynamic axis.

Let L be an initially empty linked list in which at most $O(n)$ updates in the form of insertions and deletions can be performed. We want to maintain a grouping of the elements of L in blocks of size at most s respecting the order of the elements. Further, we want to maintain an integer label of size $O(n/s)$ for each block such that labels are always strictly increasing over the blocks.

Assume first that only insertions and not deletions can be performed in L . Define $k = s/(6(1 + H_{n-1}))$ and note that $k = \Theta(s/\log n)$. When a block contains $\lceil 4k(1 + H_{n-1}) \rceil$ elements, we *mark* the middle position of the block. For every k th insertion in L we take (in constant time) a block with most elements, and if it has $5k(1 + H_{n-1})$ elements or more, we split it into two at the marked position. Lemma 11 gives that no block becomes larger than s . We use Theorem 9 to assign labels to the blocks. Each block split requires $O(\log^2 n)$ relabelings and we distribute this work over the k insertions until the next split. It can be seen that we have to relabel a block for every $\Omega(k/\log^2 n) = \Omega(s/\log^3 n)$ insertion in L . We note that from the time a position in a block is marked until the block is split, $\Omega(s)$ insertions are performed in the block. This allows us to move all existing elements of the block into the two new blocks and possibly insert them into some data structure maintained in each block. This includes maintaining pointers between a block and the elements it contains. We finally describe how to handle deletions of elements in L . We maintain a list L' which has L as a sublist. We never delete elements from L' , but when we insert an element in L we also insert it in L' such that L remains a sublist of L' . We then use the algorithm just described to maintain a block division of L' which induces the wanted block division on L (some blocks in L may become empty). The following lemma gives central properties for the list block balancing technique.

LEMMA 12. *Let L be an initially empty linked list in which at most $O(n)$ updates in the form of insertions and deletions can be performed. If we use list block balancing*

with block size $s \geq \log^3 n$ on L , then we have the following:

1. Each block has size at most s .
2. Each block has a label of size $O(n/s)$. Further, $\Omega(s/\log^3 n)$ insertions are performed in L between each relabeling of a block.
3. At most one position in each block is marked, and when a block is split it is at the marked position.
4. At least $\Omega(s)$ insertions are performed in a block from the time it is marked until it is split.

5.6. A variant of the online list labeling problem. In this section we consider a variant of the online list labeling problem defined in section 5.4 where the elements are relabeled in subconstant time per element. This is the basis for the steps O2 and O6 in section 3.2 where we convert a static axis to a dynamic axis by relabeling coordinates. In the variant we consider, the user must assign to each element $e \in L$ a unique id of $O(\log n)$ bits when e is inserted into L , and this id cannot be changed as long as e remains in L . Further, we do not require the labels to be explicitly stored in the elements of L . Instead, we require that we can calculate the label of a given element in constant time. A similar approach was used by Dietz and Sleator [21] in their proof of Theorem 8. Finally, when an element is inserted into or deleted from L , the user must be given an array of triples describing the relabeling taking place because of the insertion or deletion. A triple (i, l, l') in this array means that the element with id i and current label l is given label l' . For technical reasons we allow only a total of $O(n)$ updates in the form of insertions and deletions to be performed in L .

When n is sufficiently small compared to N (that is, if we have a sufficiently large word size and can use a sufficiently large global lookup table), the following lemma gives a way to perform the relabeling caused by an insertion or deletion in constant time. In order to be able to do this, we pack the array describing the relabeling into a single word.

LEMMA 13. *If $\log^4 n \leq \log^{1-\epsilon} N$ for a constant $0 < \epsilon < 1$, the described variant of the online list labeling problem can be solved in constant time per insertion in and deletion from L , with a relabeling cost of $O(\log^3 n)$ and a space usage of $O(n \log n)$ bits.*

Proof. We use list block balancing with block size $\log^3 n$ on L . The label assigned to an element $e \in L$ in block b consists of the label of b multiplied with $\log^3 n$ plus the rank e has in b . This assignment of labels clearly fulfills (2) in section 5.4, and because blocks have labels of size $O(n/\log^3 n)$ the labels assigned to elements in L have size $O(n)$. Further, we need only to relabel $O(1)$ blocks for each insertion in L , and since blocks have size at most $\log^3 n$ we get a relabeling cost of $O(\log^3 n)$ as claimed.

For each block we maintain an array with the ids of the elements in the block in order. We store this array in a single word which is possible because it takes up only $O(\log^4 n)$ bits. These arrays can be used to find the label of a given element in constant time using a global lookup table. Also, using these arrays we can find the array of triples in constant time to return to the user because of an update. \square

5.7. The colored predecessor problem. In this section we consider the colored predecessor problem which we define as follows. Let L be a linked list with $O(n)$ elements into which new elements can be inserted and from which existing elements can be deleted. Further, the user must assign a color to an element when it is inserted into L , and the user can also change the color of an element in L . Given an element

$e \in L$ and a color c , we must be able to find the predecessor of e in L with color c . Dietz and Raman [20, Lemma 4.2] have shown the following theorem.

THEOREM 14. *With two colors the colored predecessor problem can be solved in time $O(\log \log n)$ for updates and queries and space $O(n \log n)$ bits.*

Using an extension of the techniques in [20] we show the following theorem, where we will use only the deterministic result. The theorem may be of independent interest, which is why we also provide the randomized result.

THEOREM 15. *The colored predecessor problem without restriction on the number of colors can be solved in space $O(n \log n)$ bits and in time $O(\log^2 \log n)$ for updates and queries. Alternatively, if we allow randomization, the time usage can be reduced to $O(\log \log n)$ and the update time is then with probability $1 - n^{-c}$ for any desired constant c .*

Proof. We first describe the randomized solution. The solution builds on ideas from the list block balancing technique. We first assume we can only update L by inserting a new element. Define $k = \log^2 n$. For each color c we divide the elements of L with color c into blocks with color c containing at most $6k(1 + H_{n-1})$ elements respecting the order of the elements. We store the elements of each block in a balanced binary search tree. We mark the first element of each block b with color c except if b is the first block in L with color c . We insert all elements of L into the structure of Theorem 14 using marked and not-marked as the two colors. We use Theorem 9 to assign a label $e.\text{label}$ to each marked element $e \in L$. We create a predecessor structure P_c of Lemma 5 for each color c , and each marked element $e \in L$ with color c is inserted into P_c at position $e.\text{label}$.

Answering queries: Suppose we want to find the predecessor of $e \in L$ with color c . We first use time $O(\log \log n)$ to find the marked predecessor e' of e in L . Assuming first that e' exists, we use $e'.\text{label}$ to perform a predecessor query in P_c which will identify a block b in time $O(\log \log n)$. If e' or b does not exist, we take b to be the first block in L with color c . The block b will contain the element we are looking for (provided it exists), and this element can then be found in time $O(\log \log n)$ using the binary search tree with the elements of b . We need to be able to compare two elements in L in constant time to do this, but this can be done using Theorem 8. We conclude that the total query time becomes $O(\log \log n)$.

Inserting an element e with color c in L : First, by performing a query as just described, we identify the strict predecessor e' of e with color c . We then insert e in the block containing e' . If e' does not exist, we insert e into the first block with color c . So far, we have used $O(\log \log n)$ time. We observe that e' will never become the first element of a nonfirst block with color c ; so far we do not need to remark elements in L because of the insertion. We now describe how to ensure that blocks do not become too big. Every time we have inserted k elements in L , we take (in constant time) a largest block, and if it has at least $5k(1 + H_{n-1})$ elements we split it in $O(\log \log n)$ time into two blocks, each with at least $2k(1 + H_{n-1})$ elements. Lemma 11 ensures that no block becomes too big. Splitting a block will mark a new element of L , and thus the structure of Theorem 9 requires us to change the label of $O(\log^2 n)$ marked elements. We distribute this work over the following $k = \log^2 n$ insertions in L performing $O(1)$ relabelings on each insertion. When relabeling a marked element with color c' we also update the structure $P_{c'}$ to reflect the new label of the element. Lemma 5 then gives that we get an update time of $O(\log \log n)$ with probability $1 - n^{-c}$ for any desired constant $c > 0$.

We now support that the user can *dot* an element $e \in L$ in time $O(\log \log n)$. When answering queries we must then ignore the dotted elements. We modify the

binary search tree in each block such that we can identify the predecessor of an element among the nondotted elements. This can be done by writing in each node if the subtree rooted in that node contains a nondotted element. Next, for each color c we modify the structure P_c such that we can perform a query among the blocks which contain at least one nondotted element. Using these modified structures we can answer queries as before with one remark. When we have searched in a block b we may discover that the predecessor e' of the query is contained in the predecessor block b' of b which contain at least one nondotted element. But we can find b' and then e' in time $O(\log \log n)$.

In order to make the data structure for L we do as in the list block balancing technique. We maintain a list L' using the data structure just described such that the nondotted elements of L' correspond to the elements of L in the same order. We can then answer queries in L by asking in L' . When we insert an element in L we also insert a corresponding element in L' . When we delete an element from L we dot the corresponding element in L' . When we change the color of an element in L we dot the corresponding element in L' and insert a new corresponding element with the new color in L' . Using global rebuilding [34] we can ensure that no more than $O(n)$ updates are performed in L' .

Finally, for the deterministic result, we just use a BFAT of Theorem 6 as P_c instead of the predecessor structure of Lemma 5. \square

For readers familiar with dynamic fractional cascading [17, 29, 20] we note that we will use Theorem 15 in situations where dynamic fractional cascading has traditionally been used. The reason for this is that if we use dynamic fractional cascading, the catalog graph will get high degree, leading to bad performance in the known data structures for dynamic fractional cascading.

6. Structures supporting 3-sided queries. The main purpose of this section is to perform the steps O1 to O4 in section 3.2, thus giving our final 3-sided range reporting structure. The data structure we develop will be based on the priority search tree of McCreight [27] and on extensions of the tree similar to the ones described by Willard [45] and Arge, Samoladas, and Vitter [6]. This is combined with the use of buffers in the style of Brodal [13].

This section is organized as follows. In section 6.1 we review priority search trees with the extensions of [45, 6]. In section 6.2 we describe a dictionary for points and in section 6.3 we describe how this dictionary can be transformed into a structure similar to the priority search tree of section 6.1. In section 6.4 we describe a way to implement the structure in section 6.3 for few points giving the structure of step O1. Also, we give this structure a dynamic y-axis performing step O2. In section 6.5 we look at 3-sided structures with many points first performing step O3 and finally step O4.

6.1. Priority search trees. The following theorem is essentially shown in [27].

THEOREM 16. *There exists a structure with type $R^3(d:n, d:n)$ and performance $(\log n, \log n, n \log n)$.*

The structure of the theorem is called a priority search tree. We will not show the theorem here. Instead, we will describe a way to make a structure R with type $R^3(s:n, s:n)$ or $R^3(s:n, d:n)$, and we will also refer to this structure as a priority search tree. The structure uses ideas from [27, 45, 6], where more details can be found. We defer the exact implementation details to later sections. We span a tree T with degree $d \geq 2$ and thus height $O(\log n / \log d)$ over $R.L_x$. For each node $v \in T$ we maintain a set $v.P$ such that the following invariants are fulfilled.

- j1. The sets $v.P$ for $v \in T$ are disjoint and their union are the points of R .
- j2. If $v \in T$ is a nonroot ancestor of $w \in T$ and $w.P$ is nonempty, then $v.P$ is nonempty.
- j3. If $v \in T$ is an ancestor of $w \in T$ and $p_w \in w.P$ and $p_v \in v.P$, then $p_v.y \leq p_w.y$.
- j4. If $v \in T$ and $p \in v.P$, then v is an ancestor of the leaf with $p.x$.

Observe that given the points of R and the values $|v.P|$ for all $v \in T$ there is at most one way to select $v.P$ for $v \in T$. We define for each internal node $v \in T$ the set $v.C$ as $\cup_{w \text{ child of } v} w.P$. Suppose we are given a query $(x_1, x_2, 0, y_2)$, where 0 is the first element in $R.L_y$. Let $M \subseteq T$ be the set of internal nodes which are an ancestor to a leaf with x_1 or x_2 and note that $|M| = O(\log n / \log d)$. The query is answered as follows:

- q1. For the root $v \in T$ report the points from $v.P$ inside the query region.
- q2. Set $M' = M$.
- q3. While M' is not empty, do the following:
 - q3.1. Remove a node v from M' .
 - q3.2. Report the points from $v.C$ which are inside the query region.
 - q3.3. Add to M' the internal nodes $u \in T \setminus M$ for which a point from $u.P$ was reported in item q3.2.

LEMMA 17. *We have the following:*

- 1. *All points of R inside the query region are reported.*
- 2. *All the descendant leaves to the nodes added to M' in item q3.3 are between x_1 and x_2 on $R.L_x$.*
- 3. *If item q1. and q3.2 can be handled in time $O(1 + r)$ when r points are reported, then the query time in R becomes $O(\log n / \log d)$.*

We now describe how to update R . We insert a point p into R by inserting it into $v.P$ for the root v of T . If this insertion breaks invariant j3 or if $v.P$ has become too big (according to some additional requirements we may have imposed), we take the point with the largest y-coordinate in $v.P$, remove it from $v.P$, and insert it recursively into $w.P$ for the child w of v such that invariant j4 is preserved. We delete a point by deleting it from the set $v.P$ in which it is located. If this deletion breaks invariant j2 or if $v.P$ becomes too small (again, according to some additional requirements), we take the point p' from $v.C$ with minimal y-coordinate, add p' to $v.P$, and recursively delete p' from the set $w.P$ in which it is located (w will be a child of v).

6.2. A dictionary for points. In this section we describe a dictionary for points. Besides supporting insertion of new points, it supports deletion and lookups of existing points given their x-coordinate. The dictionary uses ideas from [13]. In section 6.2 we modify the dictionary to support 3-sided queries in the style of section 6.1. We require updates to be performed such that two different points which are in the dictionary at the same time do not have the same x-coordinate. Further, it is not legal to perform a delete operation on a point which is not in the dictionary.

Our dictionary is parametrized over three parameters $d \geq 8$, g , and r , where $g \geq r$. We create a WBB tree T with branching parameter d and leaf parameter g . We keep pebbles in the nodes of T . Each pebble represents a point and there exists three kinds of pebbles: light pebbles, insert pebbles, and delete pebbles. Light pebbles are green, while insert and delete pebbles are red. We think of light pebbles as representing contained points and of red pebbles as representing updates which are not fully completed. The internal nodes can contain only red pebbles and leaf nodes can contain only green pebbles. It is important to distinguish between pebbles

and elements in the WBB tree—the former are used to describe which points the dictionary contains, while the latter are used to balance the tree and to maintain keys to guide the search in internal nodes. For $h \geq 1$ define

$$\begin{aligned} m_h &= r \left((d-2)/(d-1) + \sum_{n=h}^{\infty} 1/d^n \right) \\ &= r(d^h - 2d^{h-1} + 1)/(d^h - d^{h-1}). \end{aligned}$$

Because $h \geq 1$ we have $m_h \leq r$. The following procedure describes how to perform $r/3$ updates to the dictionary where pebbles describing the updates are added in item u3. In this section we do not add any pebbles in item u2 or item u4.1, but we allow it to be done to support our modifications in section 6.3.

- u1. Set v to the root of T and h to the height of v .
- u2. Add at most r/d^h red pebbles to v .
- u3. Add at most $r/3$ red pebbles to v representing the updates.
- u4. While v is not a leaf, do the following:
 - u4.1. Add at most r/d^h red pebbles to v .
 - u4.2. Take m_h red pebbles from v (or as many as there is if there is fewer) and move them (one at a time) to the children of v as determined by their x-coordinate and the keys to guide the search in v .
 - u4.3. Set v to the child of v which has most red pebbles and set $h = h - 1$.

Note that in case there is more than m_h red pebbles in v in item u4.2 we do not specify which pebbles to take. When a pebble enters a node $v \in T$ we apply the following items exhaustively:

- a1. If v contains a delete pebble p and an insert pebble p' such that p deletes p' , then remove both p and p' .
- a2. If v is a leaf and contains an insert pebble p , then (1) convert p to a light pebble and (2) if there is no element with the same x-coordinate as p in the WBB tree, insert such an element into v .
- a3. If v is a leaf and contains a delete pebble p , then remove the light pebble in v that p deletes (which we know is there).

When a node v is split the red and green pebbles are distributed to the two new nodes as determined by their x-coordinate and the marked key in v (if v is an internal node) or by their x-coordinate and the elements in v (if v is a leaf). Note that we can perform a lookup in the dictionary by looking at the pebbles in the nodes between a leaf and the root of T . We have the following lemma.

LEMMA 18. *At item u1, a maximal set of siblings at height $h \geq 1$ contains at most $(d-1)m_h$ red pebbles.*

Proof. First, observe that we can ignore splitting of nodes since they do not increase the number of pebbles in a maximal set of siblings. Further, there is no problem when a new root is created because it will contain no red pebbles. Consider the root node and assume it is not a leaf. Using $d \geq 8$ it is easy to show that $r/3 + 2r/d^h \leq m_h$, giving that all pebbles added to the root in items u2, u3, and u4.1 are removed again in item u4.2. Finally, consider a maximal set S of nonroot nonleaf siblings with height h . Assume by induction that the nodes in S contain at most $(d-1)m_h$ red pebbles. Assume further that they receive at most m_{h+1} pebbles from their common parent in item u4.2 and then receive additional, at most r/d^h , pebbles in item u4.1. The total number of red pebbles added to the nodes in S is then at most $m_{h+1} + r/d^h = m_h$. Let $s \leq dm_h$ be the total number of pebbles in the

nodes of S after this and let v be the node in S with most red pebbles selected in item u4.3. There must be at least s/d pebbles in v which are then removed in item u4.2. After this the nodes in S contain at most $s(1 - 1/d) \leq dm_h(1 - 1/d) = (d - 1)m_h$ red pebbles. \square

The following lemma gives central properties for the maintained structure.

LEMMA 19. *We have the following:*

1. *The execution of items u1 to u4 inserts at most r elements in T and this is done in a set of leaves which are siblings.*
2. *The space usage of T and its contained pebbles are linear in the number of insertions.*
3. *A leaf never contains more than $2g$ green pebbles.*
4. *A set of siblings never contains more than dr red pebbles.*

Proof. Items 1 and 2 are immediate. Item 3 follows from the fact that in each leaf there is an injective mapping from the green pebbles in the leaf to the elements of the WBB tree in the leaf. Finally, item 4 is true for leaf nodes since they contain no red pebbles. For internal nodes, item 4 is just a weak version of Lemma 18 because $m_h \leq r$. \square

6.3. A structure supporting 3-sided queries. We now make a number of modifications to the structure of section 6.2 in order to make it support 3-sided queries in the style of section 6.1. Initially, we will ignore marking of keys and splitting of nodes in T . As in section 6.2 two different points in the structure at the same time cannot have the same x-coordinate and in this section they cannot have the same y-coordinate either. We will still use insert and delete pebbles to represent updates which have not been fully completed. Also, the leaf nodes can still contain only light pebbles. However, we will now store both green and red pebbles in the internal nodes of T such that the green pebbles constitute a priority search tree of section 6.1. We will keep the following invariant which corresponds to invariant j3.

- i1. *The y-coordinate of a green pebble in a node v may not be larger than the y-coordinate of any pebble below v in T .*

For each internal node $v \in T$ we define a number $igreen(v)$ which we think of as the number of green pebbles we would like to be in v (there may be less but not more). If the root $v \in T$ is not a leaf, we require

$$(3) \quad 0 \leq igreen(v) \leq 2g$$

and for other internal nodes for $v \in T$ we require

$$(4) \quad g \leq igreen(v) \leq 2g.$$

As described below, a green pebble may be deleted by a red pebble, thus bringing the number of green pebbles in v down below $igreen(v)$. For this reason we introduce a new red pebble called a *poll* pebble which basically says that there should be a green pebble which is not there. Whenever a poll pebble is moved from a node v to one of its children w in step u4.2, we move a green pebble from w to v . This is made more precise below. We maintain the following invariant.

- i2. *Suppose v is an internal node and let v_o denote the number of poll pebbles and v_g the number of green pebbles in v . Then we require $igreen(v) = v_g + v_o$. It may happen that we have no light pebble to move from w to v because all light pebbles in that part of the tree have either been deleted or moved up higher in the tree. For this reason we also introduce a new green pebble which we call a *heavy**

pebble. A heavy pebble has no x-coordinate but has a y-coordinate larger than all nonheavy pebbles. According to invariant i1, a heavy pebble will never be above a light pebble in the tree; thus the names heavy and light pebble. When we consider splitting of nodes in T below, we relax invariant i1 such that this is not always true.

We now describe what happens when a red pebble enters a node $v \in T$ in item u4.2. We apply the following items exhaustively in addition to items a1 and a2 above (items a5 and a6 make item a3 superficial):

- a4. Let p be an insert pebble in v and let p' be a green pebble in v with the largest y-coordinate among the green pebbles in v . If p has a smaller y-coordinate than p' , then (1) convert p to a light pebble; (2) if p' is a light pebble, convert it to an insert pebble; and (3) if p' is a heavy pebble, remove it.
- a5. If v contains a delete pebble p and a light pebble p' such that p deletes p' , then remove p and p' and add a poll pebble to v .
- a6. If v is a leaf, remove any poll pebble from v .

The following item describes how a poll pebble p in a node v is moved to a child of v in item u4.2. This does not follow from the description so far because poll pebbles have no x-coordinate.

- m1. Let p' be the green pebble with the smallest y-coordinate among the green pebbles stored in the children of v if it exists. If p' exists, interchange p and p' . If p' does not exist, add a heavy pebble to v and remove p .

We note that items a4, a5, a6, and m1 never increase the number of red pebbles in a node. Thus, the described pebble game is indeed just a variant of the game of section 6.2 and all the lemmas of section 6.2 still hold. Also it can be checked that invariants i1 and i2 are never broken.

The described structure can answer 3-sided queries as in section 6.1 in the following sense. In items q1 and q3.2 we report the points represented by the insert and light pebbles in the node. However, we do not report a point which is deleted by a delete pebble above it. We can avoid this as follows. First, we implement M' as a stack so that T is traversed in depth-first order. Next, while we traverse T we maintain the set of delete pebbles contained in the nodes which are ancestors to the node we are currently visiting. We then report a point only if it is not in this set. In order for the algorithm to work, we must maintain some variant of invariant j2. Essentially this is done by ensuring that all internal nonroot nodes $v \in T$ contain at least one green pebble which is not deleted by a delete pebble above it. We handle this issue in detail in section 6.4.

We now describe how to handle marking of keys and splitting of nodes in T and how to select $igreen(v)$ for internal nodes $v \in T$. If v does not have a marked key, we select $igreen(v) = 0$ if v is the root and $igreen(v) = g$ otherwise. Assume for the rest of this paragraph that v is an internal node with a marked key. Each time we pass through v in item u2 or item u4.1 we increase the value of $igreen(v)$ with $\min(2g - igreen(v), r/d^h)$ and we add the same amount of poll pebbles to v to preserve invariant i2. Since each pass through v adds at most r elements to T below v (item 1 of Lemma 19) it follows from property 6 of Lemma 7 that when v is split into v' and v'' , we have $igreen(v) = 2g$. We then set $igreen(v') = igreen(v'') = g$. In the following we modify our structure such that invariant i2 and a relaxed version of invariant i1 are preserved in v' and v'' . We will say a child of v is a child of the left (resp., right) side of v if the keys of the child are all smaller (resp., larger) than the marked key in v . Similarly, we will say a pebble is in the left (resp., right) side of v if it has an x-coordinate and if this coordinate is smaller (resp., larger) than the marked key in v . We will put each heavy pebble in the left or right side of v , and this determines

where the pebble should go when v splits. We will replace invariant i1 with invariant i1' and i1". These variants are weaker than invariant i1. Also, we will introduce a new invariant, i3.

- i1'. The y -coordinates of the g green pebbles (or all of them if there are less than g green pebbles) in a nonroot node w with smallest y -coordinates may not be larger than the y -coordinate of any pebble below w in T .
- i1". If an internal node v has a marked key, the y -coordinate of a green pebble in the left (resp., right) side of v must not be larger than the y -coordinate of any pebble below the left (resp., right) side of v .
- i3. If an internal node v has a marked key, there can be at most g green pebbles in each side of v .

Invariant i1' is sufficient to ensure we can use the same query algorithm as outlined above. Further, it can be checked that invariants i1" and i3 allow us to preserve invariant i1' when we split a node. Inspired by invariant i3 we will say the left (resp., right) side of v is *full* if v has a marked key and the left (resp., right) side of v has g green pebbles.

We now modify the way we maintain our structure in order to preserve these modified invariants. First, we modify the way we select p' in item m1 as follows. If a side of v is full, we select p' to be the green pebble with the smallest y -coordinate among the green pebbles stored in the children of the nonfull side of v . Further, if we add a heavy pebble in item m1, we do it to a nonfull side. Next, we modify the way we select p' in item a4 as follows. If p is on a full side of v , we let p' be a green pebble in v with the largest y -coordinate among the green pebbles in v on the same side of v as p . With these modifications it can be checked that the modified invariants are preserved.

6.4. Few points. The proof of the following lemma gives a way to implement the structure from section 6.3 if there are few points. This is the structure from step O1 in section 3.2.

LEMMA 20. *Suppose $d \geq 8$ and that $1 \leq t \leq (\log^{1-\epsilon} N)/(d^2 \log^3 u)$ for a constant $0 < \epsilon < 1$. Then there exists a structure with type $R^3(s : u, s : u)$ and performance $(1, 1 + \log u / \log d, u \log u, t)$.*

Proof. As a basis for the structure we use the tree T from section 6.3 with the same value of d as in the lemma here.

We store as a part of each insert and light pebble the point it represents including x - and y -coordinates using $O(\log u)$ bits for each pebble. Next, we store as a part of each delete pebble the coordinates of the point it deletes, again using $O(\log u)$ bits. Poll and heavy pebbles do not have any associated information, so any pebble can be stored using $O(\log u)$ bits. Lemma 19 item 2 now gives that the space usage of T is linear in the number of insertions performed. Using global rebuilding of [34] we can get space usage $O(u \log u)$ bits and, further, Lemma 7 property 3 gives that we can keep the height of T on at most $\min(2 \log u, O(\log u / \log d))$ (we use $2 \log u$ instead of $\log u$ because we use global rebuilding on T).

In order to get constant update time, we modify the structure as follows. We maintain a set U of insert and delete pebbles. The user can then perform t updates by adding t pebbles to U (using a global lookup table to convert them to the appropriate format). Each time the user does this we execute a constant number of steps in item u4. In item u3 we take all pebbles in U and insert them into the root of T . Since T has height $O(\log u)$ there are not too many pebbles in U if

$$(5) \quad t \log u \leq r.$$

We will select g and r such that the following properties are fulfilled.

- k1. All pebbles in a maximal set of siblings of T can be stored in $O(\log^{1-\epsilon} N)$ bits in a single word.
- k2. The maximal size of U plus the maximal number of red pebbles on a root path is larger than the minimal number of green pebbles in an internal node fulfilling invariant i1'.

Property k1 together with a global lookup table of size $2^{\log^{1-\epsilon} N}$ allows us to perform item u4.2, item u4.3, and the corresponding updates in the involved nodes in constant time. Updating T will take constant time since we have just argued it consists of a constant number of steps, each taking constant time. Note that in order for this to work, it is central that we explicitly store the x - and y -coordinates as a part of the insert, delete, and light pebbles. Property k2 together with invariant i1' implies that any internal node v contains a green pebble p which is not deleted by a delete pebble above it and such that all green pebbles in nodes descendant to v have a y -coordinate not smaller than p (p may be a heavy pebble). This essentially ensures that invariant j2 is fulfilled. More precisely it ensures that the query algorithm outlined in section 6.3 reports all points inside the query region. Also, using a global lookup table with size $2^{\log^{1-\epsilon} N}$ we can execute items q1 and q3.2 in time $O(1+r)$ when r points are reported. Lemma 17 item 3 then gives that we get query time $O(\log u / \log d)$ in T .

We now discuss how to select g in order to fulfill property k1. Because $g \geq r$, Lemma 19 items 3 and 4 together with (3) and (4) implies that a set of siblings never contains more than $3dg$ pebbles. Since each pebble uses $O(\log u)$ bits, property k1 is fulfilled if $dg \log u \leq \log^{1-\epsilon} N$. So we select

$$(6) \quad g = (\log^{1-\epsilon} N) / (d \log u).$$

We then describe how to select r in order to fulfill property k2. Using Lemma 19 item 4 the number of green pebbles in an internal node is at least $g - dr$. If we also use the fact that T has height at most $3 \log u$ (it also has height at most $2 \log u$, but we use $3 \log u$ to support the modifications in the proof of Lemma 25), the number of red pebbles on a root path plus the number of pebbles in U is at most $r + 3dr \log u$. So we require $r + 3dr \log u < g - dr$ which is fulfilled if $4dr \log u \leq g$. So based on (6) we select

$$(7) \quad r = (\log^{1-\epsilon} N) / (4d^2 \log^2 u).$$

Equation (5) together with (7) now gives our requirement for t . \square

The reader may observe that a tree with fixed shape instead of the more complicated WBB tree appears to be sufficient in order to prove Lemma 20. The reason we use a WBB tree is the proof of Lemma 30.

In order to perform step O2 in section 3.2 we need to give the structure of Lemma 20 a dynamic y -axis. Since the same construction is needed in step O7 we prove the following general lemma which can do this transformation.

LEMMA 21. *Suppose $\log^4 u \leq \log^{1-\epsilon} N$ for a constant $0 < \epsilon < 1$ and there exists a structure with type $R^\tau(s : u, s : u)$ (resp., $R^\tau(s' : u, s : u)$) and performance $(U, Q, S, \log^3 u)$. Then there exists a structure with type $R^\tau(s : u, d : u)$ (resp., $R^\tau(s' : u, d : u)$) and performance (U, Q, S) .*

Proof. We describe a solution in which at most $O(u)$ updates can be performed. This restriction can be removed by the use of global rebuilding [34]. Let R be the structure with type $R^\tau(s : u, d : u)$ (resp., $R^\tau(s' : u, d : u)$) we want to design from the structure R' with type $R^\tau(s : u, s : u)$ (resp., $R^\tau(s' : u, s : u)$). We assume each

element $e \in R$ has a unique id $e.id$ of size $O(u)$ which we must report when reporting an element from a query in R . We maintain a labeling of the elements of $R.L_y$ using Lemma 13 getting the requirement $\log^4 u \leq \log^{1-\epsilon} N$. If $e \in R$, then as id for $e.y \in R.L_y$ we use the pair $(e.id, e.x)$. Lemma 13 then assigns a label $y.label$ of size at most $O(u)$ to each $y \in R.L_y$. For each element $y \in R.L_y$ with id (i, x) we keep an element in R' with y-coordinate $y.label$, x-coordinate x , and id i . This can be done by using a global lookup table to convert the array of triples provided by Lemma 13 into updates to give to R' . A query $(x_1, x_2, y_1, y_2) \in R$ is then answered by performing the query $(x_1, x_2, y_1.label, y_2.label)$ in R' . \square

We remark that we cannot use the same technique to give the structure two dynamic axes. We finally get the structure of step O2 in section 3.2.

LEMMA 22. *Suppose $d \geq 8$ and that $\log^6 u \leq (\log^{1-\epsilon} N)/d^2$ for a constant $0 < \epsilon < 1$. Then there exists a structure with type $R^3(s : u, d : u)$ and performance $(1, 1 + \log u / \log d, u \log u)$.*

Proof. Selecting $t = \log^3 u$ in Lemma 20 gives a structure with type $R^3(s : u, s : u)$ and performance $(1, \log u / \log d, u \log u, \log^3 u)$. Further, we get the requirement that $\log^3 u \leq (\log^{1-\epsilon} N)/(d^2 \log^3 u)$ or, equivalently, $\log^6 u \leq (\log^{1-\epsilon} N)/d^2$. Finally, inserting the obtained structure into Lemma 21 gives the desired result since the requirement $\log^4 u \leq \log^{1-\epsilon} N$ is fulfilled by our other requirement for u . \square

6.5. Many points. The following lemma gives our structure from step O4.

LEMMA 23. *For any sufficiently small constant $\epsilon > 0$ there exists a structure with type $R^3(s : n, d : n)$ and performance $(1 + \log n / \log^{1/6-\epsilon} N, 1 + \log n / \log \log N, n \log n)$.*

Proof. Define $u = \min(n, 2^{\log^{1/6-\epsilon/2} N})$. We let R' be the structure we get from Lemma 22 if we set d in the lemma to $\log^\epsilon N$. Then the maximal u we can use in the lemma is equal to the u we have just defined and R' gets type $R^3(s : u, d : u)$ and performance $(1, 1 + \log u / \log \log N, u \log u)$.

We assume for the rest of the proof that $n \geq u$ and thus $u = 2^{\log^{1/6-\epsilon/2} N}$ since otherwise R' is the structure we need to prove the theorem. We let the structure R with type $R^3(s : n, d : n)$ we want to design be a priority search tree of section 6.1. We give T degree u and require $|v.P| \leq 1$ for all $v \in T$. In each internal node $v \in T$ we keep a structure $v.R$ with the same type as R' . Let $v \in T$ be an internal node and assume $p \in w.P$ for a child w of v . We then store a point $p' \in v.R$ corresponding to p with $p'.y = p.y$ and with $p'.x$ set to the number of siblings w has to the left in T . Also, we link p and p' together with pointers. For each element $e \in R.L_y$ there is a unique node $v \in T$ and a unique element $e' \in v.R.L_y$ such that e and e' represent the same y-coordinate. We make a pointer from e to e' and color e with the color v . We maintain the colored predecessor structure of Theorem 15 on $R.L_y$ using the assigned colors.

We answer queries as in section 6.1 with the following remarks. Assume first a node $v \in M \cap M'$ is picked in item q3.1. In item q3.2 we then do as follows. For the (at most two) children $w \in M \cap M'$ of v we report the point in $w.P$ if it is in the query region. The remaining points to report can be found by a query in $v.R$. In order to find the y-coordinate to use in this query we make a query for the color of v in the structure of Theorem 15 maintained on $R.L_y$. We conclude that we must use a total time of $O(\log u / \log \log N + \log^2 \log n + r)$ to report r elements from v . This is also $O(\log u / \log \log N + r)$ since $u \leq n \leq N$. Assume next a node $v \in M' \setminus M$ is picked in item q3.1. Then Lemma 17 item 2 gives that we should report all the points in $v.C$ with a sufficiently low y-coordinate. These points can be found in constant time per point by walking through $v.R.L_y$ from the beginning stopping when the y-coordinate

becomes too big. Since $|M| = O(\log n / \log u)$ we conclude that the query time in R becomes $O((\log u / \log \log N) \log n / \log u) = O(\log n / \log \log N)$.

Updates can also be performed as in section 6.1 with remarks similar to the ones described for queries. For a total of $O(\log n / \log u)$ nodes $v \in T$ we need to update $v.R$ and to recolor an element of $R.L_y$. Updating $v.R$ takes constant time, while updating $R.L_y$ takes time $O(\log^2 \log n)$. The total update time thus becomes $O(\log^2 \log n \log n / \log u) = O(\log^2 \log n \log n / \log^{1/6-\epsilon/2} N) = O(\log n / \log^{1/6-\epsilon} N)$ as claimed. \square

Finally, the following theorem gives the structure of step O4 from section 3.2. For $n = N$ this gives the second result from the introduction.

THEOREM 24. *For any sufficiently small constant $\epsilon > 0$ there exists a structure with type $R^3(d : n, d : n)$ and performance $(\log \log n + \log n / \log^{1/6-\epsilon} N, \log \log n + \log n / \log \log N, n \log n)$.*

Proof. We describe a solution in which at most $O(n)$ updates can be performed. This restriction can be removed by the use of global rebuilding [34]. Let R be the structure with type $R^3(d : n, d : n)$ we want to design. We maintain a structure T of Lemma 23 with type $R^3(s : n / \log^3 n, d : n / \log^3 n)$ and performance $(1 + \log n / \log^{1/6-\epsilon} N, 1 + \log n / \log \log N, n / \log^2 n)$. We use list block balancing with block size $\log^3 n$ on $R.L_x$. We keep the points of a block b in a structure $b.R$ of Theorem 16. Let b be a block with assigned label x and let y be the first element of $b.R.L_y$. We then insert b in T with x -coordinate x and y -coordinate y . This is possible because labels of blocks have size $O(n / \log^3 n)$. If an element $e \in R.L_y$ represents the same y -coordinate as an element $e' \in T.L_y$ (resp., $e'' \in b.R.L_y$), we link e and e' (resp., e'') together with pointers.

We maintain the structure of Theorem 14 on $R.L_y$ giving elements which have a pointer to an element in $T.L_y$ one color and other elements another color. This allows us to identify the position in $T.L_y$ of any element in $R.L_y$ in time $O(\log \log n)$. For each block b we keep the elements of $b.R.L_y$ in a balanced binary search tree and we maintain a structure of Theorem 8 on $R.L_y$. This allows us to identify the position in $b.R.L_y$ of any element in $R.L_y$ also in time $O(\log \log n)$.

We need to relabel $O(1)$ blocks for each update in R , so we need to perform $O(1)$ updates in T for each update in R . Since updating $b.R$ for a block b and identifying elements in and updating $b.R.L_y$ and $T.L_y$ takes time $O(\log \log n)$ the total update time becomes $O(\log \log n + \log n / \log^{1/6-\epsilon} N)$.

We now describe how to answer a query $(x_1, x_2, 0, y_2)$, where 0 is the minimal element of $R.L_y$. Let b_1 be the block with x_1 and b_2 be the block with x_2 . We then answer the query by first making a local query in $b_1.R$ and then a local query in $b_2.R$ using time $O(\log \log n)$ (if $b_1 = b_2$, we need only to perform one local query in total). The remaining points can be found by performing a query in T . For each reported block b we identify the points from b to report by walking through $b.R.L_y$ from the beginning stopping when the y -coordinate becomes larger than y_2 (as in the proof of Lemma 23). Identifying the elements to query from in $b_1.R.L_y$, $b_2.R.L_y$, and $T.L_y$ requires time $O(\log \log n)$, so the total query time becomes $O(\log \log n + \log n / \log \log N)$. \square

7. Structures supporting 4-sided queries. The purpose of this section is to perform steps O5 to O10 in section 3.2, thus giving our final 4-sided range reporting structure for the 2-dimensional case. The section is organized as follows. In section 7.1 we describe a simple way to make a structure supporting 4-sided queries from one supporting 3-sided queries. In section 7.2 we use this to perform steps O5, O6,

and O7. We then apply the construction from section 4 to perform step O8 and then we perform step O9. Finally, in section 7.3 we perform step O10.

7.1. From 3-sided to 4-sided. In this section we describe a simple way to make a structure supporting 4-sided queries from one supporting 3-sided queries. Similar ideas have been used by Chazelle [14] and Overmars [35]. Suppose for any m , $m \leq n$, we have a structure with type $R^3(s' : n, s : m)$ and performance (U_m, Q_m, S_m) . We will now describe how to make a structure R with type $R^4(s' : n, s : n)$ and performance $(U_n \log n, Q_n, S_n \log n)$, provided U_m and Q_m are nondecreasing functions of m and provided S_m is a subadditive function of m . We span a complete binary tree T over $R.L_y$. Each node $v \in T$ spans a subinterval I of $R.L_y$, where $|I| \leq n$. We store in v two secondary structures $v.R_1$ and $v.R_2$ both with type $R^3(s' : n, s : |I|)$ and both containing all points of R with a y -coordinate in I . The structure $v.R_1$ (resp., $v.R_2$) should support queries of the form (x_1, x_2, y_1, y_2) , where y_1 (resp., y_2) is the leftmost (resp., rightmost) element in I . Inserting (resp., deleting) a point p into (resp., from) R requires inserting (resp., deleting) p into (resp., from) two secondary structures in each of the $O(\log n)$ ancestors of the leaf of T with $p.y$ using time $O(U \log n)$. Suppose we are given a query (x_1, x_2, y_1, y_2) . Let $v \in T$ be the nearest common ancestor of y_1 and y_2 and let v_l be the left and v_r the right child of v . The query can then be answered by performing a single query in $v_l.R_2$ and a single query in $v_r.R_1$ giving a query time $O(Q)$.

7.2. At least one short axis. The following lemma gives the structure from step O5 of section 3.2.

LEMMA 25. *Suppose $1 \leq t \leq (\log^{1-\epsilon} N)/\log^4 u$ for a constant $0 < \epsilon < 1$. Then there exists a structure with type $R^4(s' : u, s : u)$ and performance $(1, 1 + \log u, u \log^2 u, t)$.*

Proof. We will first how to construct a structure with type $R^4(s : u, s : u)$ and the claimed performance. We will do this by combining the proof of Lemma 20 with the construction in section 7.1. We keep all points in the dictionary of section 6.2. To avoid confusion we call the tree T' instead of T . Further, we make a number of modifications and simplifications described in the following. First, we use the y -instead of the x -coordinates of points. Second, we let T' be a complete binary tree with fixed shape, so nodes never split and keys are never marked. This implies that each leaf contains at most one green pebble. Third, we define $m_h = r/3$ for all h . Fourth, in our procedure for performing $r/3$ updates in T' we remove items u2 and u4.1. Lemma 18 can be shown to still hold with these modifications (a simpler proof carries through).

As in section 7.1 we have two secondary structures in each internal node of T' . Each time a pebble is moved from a node $v \in T'$ in item u4.2, a copy of the pebble is also given to the two secondary structures in v . For the secondary structures, we use structures similar to the one of Lemma 20 with $d = 8$. We select the same value for r in T' as in Lemma 20, namely the one given by (7) with $d = 8$. We can now answer queries as in section 7.1. We note that when reporting points from the secondary structures in T' we must also take the delete pebbles of the relevant nodes of T' into account. But this is not a problem because the height of T' plus the height of any secondary structure is at most $3 \log u$.

As in the proof of Lemma 20 we will add only t pebbles at a time to the root of T' (where t will be determined in a moment). We note that during the time $r/3$ pebbles are inserted into the root of T' we must go through items u1 to u4 as in the proof of Lemma 20. Further, each time we execute item u4.2 in T' we must also execute

the loop in items u1 to u4 in a constant number of secondary structures. Since the height of T' as well as the height of any secondary structure is at most $O(\log u)$ the following condition ensures that we do not add too many pebbles to the root of T' before we empty it again:

$$t \log^2 u \leq r.$$

Together with (7) this gives our requirement for t .

At most $O(\log u)$ pebbles are created for each insertion in T' , so using global rebuilding [34] we can get a space usage of $O(u \log^2 u)$ bits.

We give the structure type $R^4(s':u, s:u)$ using a standard trick: We insert each point e into the structure just described as a point e' with $e'.x = c \cdot u \cdot e.x + e.y$ and $e'.y = e.y$ for a constant c . Clearly, each point inserted will have unique x- and y-coordinates. However, it doubles the number of bits in each x-coordinate. It can be checked in our construction that this is not a problem. \square

As a corollary we get the following lemma, which gives the structure from step O6 of section 3.2.

LEMMA 26. *Suppose $\log^7 u \leq \log^{1-\epsilon} N$ for a constant $0 < \epsilon < 1$. Then there exists a structure with type $R^4(d:u, s':u)$ and performance $(1, \log u, u \log^2 u)$.*

Proof. Selecting $t = \log^3 u$ in Lemma 25 gives a structure with type $R^4(s':u, s:u)$ and performance $(1, \log u, u \log^2 u, \log^3 u)$. Further, we get the restriction on u that $\log^3 u \leq (\log^{1-\epsilon} N) / \log^4 u$ or, equivalently, $\log^7 u \leq \log^{1-\epsilon} N$. Finally, inserting the obtained structure into Lemma 21 and interchanging the two axes gives the desired result since the requirement $\log^4 u \leq \log^{1-\epsilon} N$ is fulfilled by our other requirement for u . \square

As a corollary, we get the structure from step O7 of section 3.2.

LEMMA 27. *Suppose $\log^7 u \leq \log^{1-\epsilon} N$ for a constant $0 < \epsilon < 1$. Then for any $n \leq N$ there exists a structure with type $R^4(s':n, s:u)$ and performance $(\log^2 \log n, \log u + \log^2 \log n, u(\log^2 u + \log n))$.*

Proof. Let R' be the structure of Lemma 26 restricted to having type $R^4(d:u, s:u)$. Let R be the structure with type $R^4(s':n, s:u)$ we want to design. We insert each element $e \in R$ into a BFAT of Theorem 6 with universe size $O((n+1)u)$ at position $u \cdot e.x + e.y$. We link all elements in R together according to this position. This list can be used as $R'.L_x$. Further, at most u elements will be stored in the BFAT and this concludes the proof of the lemma. \square

Plugging the structure of this lemma into Lemma 2 we get the structure of step O8 of section 3.2.

LEMMA 28. *Suppose $\log^7 u \leq \log^{1-\epsilon} N$ for a constant $0 < \epsilon < 1$. Then for any $n \leq N$ there exists a structure with type $R^4(s':n, s':u)$ and performance $(\log^3 \log n, \log u \log \log n + \log^3 \log n, nu \log n \log \log n(\log^2 u + \log n))$.*

The following lemma gives the structure from step O9 of section 3.2.

LEMMA 29. *For any $n \leq N$ and for any u where $\log^3 n \leq u$ and $\log^7 u \leq \log^{1-\epsilon} N$ for a constant $0 < \epsilon < 1$ there exists a structure with type $R^4(d:n, s':u/\log^3 n)$ and performance $(\log^3 \log n, (\log u + \log^2 \log n) \log \log n, n \log^2 u + n \log n)$.*

Proof. We describe a solution in which at most $O(n)$ updates can be performed. This restriction can be removed by the use of global rebuilding [34]. Let T be the structure of Lemma 28 where we select u in the lemma as $u/\log^3 n$ and n in the lemma as n/u . The structure T then gets type $R^4(s':n/u, s':u/\log^3 n)$ and performance $(\log^3 \log n, \log u \log \log n + \log^3 \log n, n \log n)$ (the space usage is actually smaller).

Let R be the structure with type $R^4(d : n, s' : u / \log^3 n)$ we want to design. We use list block balancing with block size u on $R.L_x$. If a block b with assigned label x contains one or more points with y -coordinate y , we insert b in T at x -coordinate x and y -coordinate y . This is possible because labels have size $O(n/u)$. Further, we need only to relabel a block for every $O(u / \log^3 n)$ update of R . Since this is also the length of $R.L_y$ we need only to perform $O(1)$ updates in T for each update in R . We maintain in each block b a structure $b.R$ of Lemma 26 with type $R^4(d : u, s' : u)$ and performance $(1, \log u, u \log^2 u)$. All points in b are stored in $b.R$ and the total space used by these structures is $O(n \log^2 u + n \log n)$ bits.

Suppose we are given a query (x_1, x_2, y_1, y_2) . Let b_1 be the block with x_1 and b_2 be the block with x_2 . We answer the query by first performing a local query in $b_1.R$ and then a local query in $b_2.R$ (if $b_1 = b_2$, we need only to perform one local query in total). The remaining points can be found by performing a query in T and then reporting relevant points from the reported blocks. This can be done in constant time per point if we for each block b and each y -coordinate $y \in R.L_y$ maintain a linked list with the points in b with y -coordinate y .

The space usage of the structure is dominated by the space used in the $b.R$ structures for the different blocks b . The update time is dominated by the time used in T and this concludes the proof of the lemma. \square

The following lemma gives the structure of Lemma 29 a somewhat dynamic y -axis.

LEMMA 30. *The structure of Lemma 29 can be given a dynamic y -axis in the following sense. The y -axis is a linked list where each element has a unique id of size $O(u)$. The id of an element is fixed when the element is inserted in the y -axis and we do not allow elements to be deleted from the y -axis. We allow many points to share a y -coordinate. We must provide a function f which, given (in a single word) an array with $O(\log^{1-\epsilon} N / \log u)$ ids, in constant time returns an array where the ids are sorted according to the order the elements with these ids have on the y -axis.*

Proof. Looking into the proofs of Lemma 27, Lemma 28, and Lemma 29 together with the construction of section 4 we see that it is sufficient to show that we can get a dynamic y -axis in the described way in the structure of Lemma 26. From the proof of this lemma it follows that it is sufficient that we can get a dynamic x -axis in the described way in Lemma 25. When inserting a point into this structure we will use the id of the x -coordinate as x -coordinate. Looking into the proof of Lemma 25 we see that the x -coordinates of points are not used in the maintenance of the tree T' . They are, however, used in the secondary structures which are the structures of Lemma 20. Here they are used as x -coordinates of pebbles and as keys of the WBB tree. However, the function f will give us all the information we need about the order of these coordinates. Further, it will do it fast enough because the construction in Lemma 20 never handles more than $O(\log^{1-\epsilon} N / \log u)$ pebbles at once. \square

7.3. Two long axes. We are now ready to prove the following theorem which for $n = N$ shows the first result from the introduction for $d = 2$.

THEOREM 31. *For any constant $\gamma > 0$ there exists a structure with type $R^4(d : n, d : n)$ and performance $(\log^{7/8+\gamma} N, \log N / \log \log N, \log^{1+7/8+\gamma} N)$.*

The structure we construct in the proof of the theorem is a kind of range tree (see section 1.3). Traditionally, range trees have degree 2. Our range trees are split into two parts. In the *top part*, nodes have degree u of Lemma 30 and we keep the points of an internal node in a structure of Lemma 30. In the *bottom part*, nodes have constant degree. The reason we need the bottom part is that the structure of Lemma 30 does not have a completely dynamic y -axis. The bottom part ensures that

nodes in the top part have sufficiently many points below them to allow the table to compute the function f of Lemma 30 to be built. Finally, to keep the query time low we keep in some levels of the tree the points in the structure of Theorem 24.

Proof of Theorem 31. Let R be the structure with type $R^4(d:n, d:n)$ we want to design. We store the elements of R in a WBB tree T with leaf parameter 1 and for $e \in R$ we use $e.y$ as key. We specify the degree of T in a moment. We let v_r denote the root of T . We first ignore splitting of nodes and marking of keys. We keep in each node $v \in T$ a linked list $v.L$ containing the points located in the leaves descendant to v sorted according to their x-coordinate. Let $v \in T$ be an internal node. We color the children of v such that no two different children of v have the same color. For each $e \in v.L$ there is exactly one child v' of v and one $e' \in v'.L$ such that e and e' represent the same point. We color e with the color of v' and we keep in e a pointer to e' (such a pointer is called a *downpointer* [42]). We keep the elements of $v.L$ in a colored predecessor structure of Theorem 15 using the assigned colors. The update and query time of this structure is $O(\log^2 \log |v.L|)$; we will take it to be $O(\log^2 \log N)$.

Suppose we are given a query (x_1, x_2, y_1, y_2) in R . We observe that $v_r.L$ contains all points of R sorted according to their x-coordinate. The coordinates x_1 and x_2 identify two elements in $v_r.L$ corresponding to the query interval on the x-axis. Further, using the pointers of the nodes of T to their parents we can identify the simple path in T between v_r and the leaf identified by y_1 . We proceed from v_r down along this path using the downpointers and the colored predecessor structures to maintain the query interval on $v.L$ for the nodes $v \in T$ we meet. Beginning in some node of T , we must report the elements in this interval that has a relevant set of colors. After this, we repeat the same process with y_2 instead y_1 and we are done.

Inserting a point p into R is performed in a way similar to the way a query is answered. $p.x$ identifies a position in $v_r.L$ where the point should be inserted. Also, $p.y$ identifies a simple path in T between the leaf with $p.y$ and v_r . We proceed from v_r down along this path inserting p into $v.L$ for the nodes v we meet and using the downpointers and the colored predecessor structures to maintain the position to insert into.

Deleting a point p from R is done as follows. $p.x$ determines the location of p in $v_r.L$. We then delete p from $v_r.L$ and use the downpointers to remove p from the other lists $v.L$ it is in. We do not delete p from T . Instead, we use global rebuilding [34] to ensure that at most half of the elements of T are deleted points.

Let $\epsilon > 0$ be a constant to be determined in a moment and suppose $v \in T$. If v has a height smaller than $\log^{1-\epsilon} N$, we say v is in the bottom part and otherwise we say v is in the top part of T . We will give v different degree parameters depending on whether it is in the bottom or top part. It is straightforward to modify WBB trees to support that. If a node w has a child v in the bottom part, we say that w is a leaf of the top part and that v is a root of the bottom part. Below we will analyze the time we need to use in the top and bottom part when performing an update or a query. Also, we will analyze how much space is used in the top and bottom part. Our results are summarized in Table 1.

TABLE 1

	Top part	Bottom part
Update time	$O(\log^{6/7+\epsilon/7} N \log^3 \log N)$	$O(\log^{1-\epsilon} N \log^2 \log N)$
Query time	$O(\log N / \log \log N)$	$O(\log^{1-\epsilon} N \log^2 \log N)$
Bits used	$O(n \log^{1+6/7+\epsilon/7} N)$	$O(n \log^{2+\epsilon} N)$

Ignoring $\log \log N$ factors the update time and space usage in the upper and lower part becomes identical if we select $\epsilon = 1/8$. The values here then become smaller than the ones claimed in the lemma.

Suppose $v \in T$ is a node in the bottom part. We then give v degree parameter 8 which is the minimal degree parameter we allow for WBB trees. We note that according to Lemma 7 property 5 a subtree rooted at a root in the bottom part of T has $\Theta(8^{\log^{1-\epsilon} N})$ leaves and according to property 3 has height $O(\log^{1-\epsilon} N)$. If we link the elements of $v.L$ with the same color together in order, we need only to spend time $O(\log^2 \log N)$ plus constant time for each point reported when we visit v during a query. Also, when visiting v during an update, we need to spend time $O(\log^2 \log N)$. We conclude that we get update and query time $O(\log^{1-\epsilon} N \log^2 \log N)$ in the bottom part of T . Since each point of R is stored in exactly one list $v.L$ for each level of T the total space usage for the bottom part becomes $O(n \log^{2-\epsilon} N)$ bits.

Suppose now $v \in T$ is a node in the top part. We store the elements of $v.L$ in the structure of Lemma 30 using the colors as ids of y-coordinates. For this reason we give v degree parameter $u = 2^{\log^{(1-\epsilon)/7} N}$ and note that with this selection the structure of Lemma 30 with an x-axis of length m gets performance $(\log^3 \log N, \log u \log \log N, m \log N)$. The maximal length of a simple path from the root of T to a leaf of the top part is $O(\log N / \log u) = O(\log^{6/7+\epsilon/7} N)$. We keep in v a table $v.G$ with length $8^{\log^{1-\epsilon} N}$. Such a table is large enough to allow f , needed by Lemma 30 to be computed in constant time. Also, it will not increase the space usage except for a constant factor. The update time in the top part becomes $O(\log^{6/7+\epsilon/7} N \log^3 \log N)$ and the space usage becomes $O((\log N / \log u) n \log N) = O(n \log^{1+6/7+\epsilon/7} N)$ bits again because each point of R is stored in exactly one list $v.L$ for each level of T .

The query time in the top part of T is $O((\log N / \log u) \log u \log \log N) = O(\log N \log \log N)$ and this is an $O(\log^2 \log N)$ factor too high. We fix this problem as follows. For each node v in the top part of T which is on a level divisible by $\log N / ((\log^2 \log N) \log u)$, we keep two structures of Theorem 24. In these structures we save the points of $v.L$ in the style of section 7.1. We then modify the way we answer queries as follows. When we in our proceeding down in T meet a node v in the top part of T on a level divisible by $\log N / ((\log^2 \log N) \log u)$ and when we have tried to report points from $w.L$ for the parent w of v , we stop. We then perform a query in one of the structures of Theorem 24 kept in v . After this, we do not need to proceed further down in T . This modification reduces the query time in the top part of T to $O(\log N / \log \log N)$. Further, since each point of R is stored in at most $O(\log^2 \log N)$ structures of Theorem 24, the time and space usage is not increased except for constant factors.

We now describe how to handle marking of keys and splitting of nodes in T . Consider the situation where a key k in a node v with parent w is marked (to simplify the discussion we assume that w exists or equivalently that v is not the root). Suppose v will eventually split into itself and a new node v' which will become the right sibling of v . We then select a color c' for v' such that no other current or coming child of w has color c' . During the time v is marked we move the points from $v.L$ with keys larger than k to $v'.L$. When a point p is moved we also color p with c' in $w.L$. Using the remark following Lemma 7 things can be adjusted such that we are finished when v split such that v can be split in constant time. We have to adjust queries such that they can handle partly split nodes. The details are tedious but standard and are thus omitted. If v contains a structure of Theorem 24, the splitting of this structure can

be handled in a way similar to the one just described. One problem remains. If v is in the top part of T we need to insert c' on the y-axis of the structure of Lemma 30 into w . This is done by updating $w.G$. Since w is in the top part of T it follows from Lemma 7 item 5 that $\Omega(2^{\log^{1-\epsilon} N})$ insertions of elements descendant to v will be performed before v splits, so we have time to rebuild $w.G$. We rebuild parts of $w.G$ during these insertions as follows (stating that all of $w.G$ should be rebuilt does not make sense because several children of w may be marked at the same time). When a key in v is marked there is a current set C of colors given to current or coming children of w . During the insertions of elements descendant to v before v splits we fill in all entries of $w.G$ which compare colors from $C \cup \{c'\}$. This will ensure that $w.G$ can always compare any set of colors of the children of w . In order to make queries work while v is splitting we actually need to update $w.G$ before we start building up $v'.L$, building up $v''.L$, and changing color of elements in $w.L$. \square

8. Higher dimensions. We will say a structure for the d -dimensional range reporting problem in the list order variant has type R_n^d if it can contain n points. Further, we will say such a structure has performance (U, Q, S) if it has update time $O(U)$, query time $O(Q)$, and space usage $O(S)$ bits. As in section 3.1 we allow U , Q , and S to depend on d and n but not on the number of points stored. This section is devoted to the proof of the following theorem which for $n = N$ shows the first result from the introduction.

THEOREM 32. *Let $d \geq 2$ and $\epsilon > 0$ be any constants and define $\omega = 7/8 + \epsilon$. Then for any $n \leq N$ there exists a structure for R_n^d with performance $(\log^{d-2+\omega} N, (\log N / \log \log N)^{d-1}, n \log^{d-1+\omega} N)$.*

Our proof of Theorem 32 is based on Lemma 33 below, which is a dynamization of a method proposed by Alstrup, Brodal, and Rauhe [3]. They described how to extend the dimension of a range reporting structure in a static setting where updates were not allowed.

LEMMA 33. *Suppose for any n we have a data structure with type R_n^d and performance (U_n, Q_n, S_n) , where U_n and Q_n are nondecreasing functions of n and S_n is a subadditive function of n . Then for any n and any constant $\epsilon > 0$ we can make a structure for R_n^{d+1} with performance $((U_n + \log^2 \log n) \log^{1+\epsilon} n, (Q_n + \log^2 \log n) \log n / \log \log n, S_n \log^{1+\epsilon} n)$.*

Proof. Let R be the structure with type R_n^{d+1} we want to make. For the proof we fix an axis of R , say, the x-axis. We store the elements of R in a WBB tree T with degree parameter $\log^{\epsilon/2} n$ and leaf parameter 1. For $e \in R$ we use $e.x$ as key, where $e.x$ is the x-coordinate of e . We first ignore deletion of points, splitting of nodes, and marking of keys.

Lemma 7 item 3 gives that T gets height $O(\log n / \log \log n)$. Let $v \in T$ be an internal node with height h and t children. For each pair (i, j) , where $1 \leq i \leq j \leq t$, we keep a secondary structure $v.R_{(i,j)}$ with the given type R_m^d , where $m = 4 \log^{h\epsilon/2} n$. Let v_1, \dots, v_t be the ordered children of v . We then in $v.R_{(i,j)}$ save all points which are in leaves descendant to v_k , where $i \leq k \leq j$ and where we ignore the x-coordinates when we store the points. Lemma 7 item 5 gives that there is room for these points in $v.R_{(i,j)}$.

We give each secondary structure of T a unique color. For any axis $R.L_y$ different from the x-axis let $v.R_{(i,j)}.L_y$ be the same axis of the secondary structure $v.R_{(i,j)}$. We then maintain pointers between the elements of $v.R_{(i,j)}.L_y$ and the corresponding elements of $R.L_y$. Each element in $R.L_y$ will contain a number of pointers and we use the BFAT of Theorem 6 (as a dictionary) to store these. Also, we maintain a structure

of Theorem 15 on $R.L_y$ where an element has the colors of the secondary structures it has pointers to (Theorem 15 can easily be generalized to allow an element to have any number of colors).

Assume we are given a query in R . By following pointers from nodes of T to their parents we can in linear time identify $O(\log n / \log \log n)$ secondary structures such that asking the query in these structures (ignoring the x-axis) gives the desired points. For each secondary structure the position on the axes in which we should perform the query can be found in time $O(\log^2 \log n)$ using the structure of Theorem 15 maintained on the axes of R . This shows that the query time becomes $O((Q_n + \log^2 \log n) \log n / \log \log n)$ as claimed (Q_n is a nondecreasing function of n).

Each point is stored in $O(\log^\epsilon n)$ secondary structures in $O(\log n / \log \log n)$ nodes of T . When inserting new points the updates to perform in the secondary structures can be found in a way similar to when answering queries and the update time becomes $O((U_n + \log^2 \log n) \log^{1+\epsilon} n / \log \log n)$ (U_n is a nondecreasing function of n). The space usage becomes $O(S_n \log^{1+\epsilon} n / \log \log n)$ bits (because S_n is a subadditive function of n).

Marking of keys and splitting of nodes can be handled as described after the remark of Lemma 7. The details are, as in the proof of Theorem 31, tedious but standard and are thus omitted. Finally, we delete a point by removing it from the axes of R and from the secondary structures in which it is located. We never delete elements from T . Instead, we use global rebuilding [34] to ensure that at most half of the elements of T are deleted points. \square

Finally, we have the following proof.

Proof of Theorem 32. If $d = 2$, Theorem 31 gives the desired result. Otherwise, we get the desired structure by applying Lemma 33 $d - 2$ times to Theorem 31 where we set $n = N$. \square

9. Open problems. We have given an optimal structure in terms of time per operation for the 2-dimensional range reporting on RAM. Also, we have given a new solution for the d -dimensional range reporting problem for any constant dimension $d \geq 3$. A very interesting open problem is to prove or disprove that our solution is optimal for any constant dimension $d \geq 3$. Showing a matching lower bound seems to be well beyond current lower-bound techniques.

The construction leading to our 2-dimensional range reporting structure (Theorem 31) is rather involved. Of course, it would be nice to have a simpler construction obtaining the same or a similar bound.

For query time $O(\log n / \log \log n)$, the update time of our 2-dimensional range reporting structures is $O(\log^\omega n)$ for a constant $\omega < 1$. However, the lower bound of [4] does not prevent even constant update time in this case. An open problem is to close or narrow the gap between these two update times.

The basic technique in the data structures of this paper has been to pack many points together in a single word. This has allowed us to process points in subconstant time per point. This idea is not new and is very similar to what is done in the I/O model of computation [2], where elements (or points) are packed together on disk blocks. For this reason, data structures for the I/O model and for the RAM model of computation have often inspired each other, and we have also used ideas from the I/O model in this paper. An open problem is to use ideas of this paper in the I/O model. In particular, it may be possible to reduce the update time of the structure of Arge, Samoladas, and Vitter [6] for dynamic range reporting in the I/O model of computation by using ideas from this paper. Such a solution may even be practical.

Acknowledgments. I am very grateful to my supervisors Stephen Alstrup and Theis Rauhe for introducing me to the range reporting problem, for bringing my attention to [13], and for helpful discussions in connection with this work. I would like to thank Dan Willard for clarifying the history of range trees. Finally, I would like to thank the anonymous referees for comments which have helped me to improve the presentation.

REFERENCES

- [1] P. K. AGARWAL AND J. ERICKSON, *Geometric range searching and its relatives*, in Advances in Discrete and Computational Geometry, AMS, Providence, RI, 1999, pp. 1–56.
- [2] A. AGGARWAL AND J. S. VITTER, *The input/output complexity of sorting and related problems*, Comm. ACM, 31 (1988), pp. 1116–1127.
- [3] S. ALSTRUP, G. S. BRODAL, AND T. RAUHE, *New data structures for orthogonal range searching*, in Proceedings of the 41st Annual Symposium on Foundations of Computer Science, IEEE Computer Society Press, Los Alamitos, CA, 2000, pp. 198–207.
- [4] S. ALSTRUP, T. HUSFELDT, AND T. RAUHE, *Marked ancestor problems*, in Proceedings of the 39th Annual Symposium on Foundations of Computer Science, IEEE Computer Society Press, Los Alamitos, CA, 1998, pp. 534–543.
- [5] A. ANDERSSON AND M. THORUP, *Tight(er) worst-case bounds on dynamic searching and priority queues*, in Proceedings of the 32nd Annual ACM Symposium on Theory of Computing, ACM, New York, 2000, pp. 335–342.
- [6] L. ARGE, V. SAMOLADAS, AND J. S. VITTER, *On two-dimensional indexability and optimal range search indexing*, in Proceedings of the 18th ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, ACM, New York, 1999, pp. 346–357.
- [7] L. ARGE AND J. S. VITTER, *Optimal external memory interval management*, SIAM J. Comput., 32 (2003), pp. 1488–1508.
- [8] P. BEAME AND F. E. FICH, *Optimal bounds for the predecessor problem*, in Proceedings of the 31st Annual ACM Symposium on Theory of Computing, ACM, New York, 1999, pp. 295–304.
- [9] J. L. BENTLEY, *Decomposable searching problems*, Inform. Process. Lett., 8 (1979), pp. 244–251.
- [10] J. L. BENTLEY, *Multidimensional divide-and-conquer*, Comm. ACM, 23 (1980), pp. 214–229.
- [11] J. L. BENTLEY AND J. B. SAXE, *Decomposable searching problems. I. Static-to-dynamic transformation*, J. Algorithms, 1 (1980), pp. 301–358.
- [12] J. L. BENTLEY AND M. I. SHAMOS, *A problem in multivariate statistics: Algorithm, data structure, and applications*, in Proceedings of the Fifteenth Allerton Conference on Communication, Control, and Computing, 1977, pp. 193–201.
- [13] G. S. BRODAL, *Predecessor queries in dynamic integer sets*, in Proceedings of the 14th Annual Symposium on Theoretical Aspects of Computer Science, Lecture Notes in Comput. Sci. 1200, Springer, Berlin, 1997, pp. 21–32.
- [14] B. CHAZELLE, *Filtering search: A new approach to query-answering*, SIAM J. Comput., 15 (1986), pp. 703–724.
- [15] B. CHAZELLE, *A functional approach to data structures and its use in multidimensional searching*, SIAM J. Comput., 17 (1988), pp. 427–462.
- [16] B. CHAZELLE, *Lower bounds for orthogonal range searching: I. The reporting case*, J. ACM, 37 (1990), pp. 200–212.
- [17] B. CHAZELLE AND L. J. GUIBAS, *Fractional cascading: I. A data structuring technique*, Algorithmica, 1 (1986), pp. 133–162.
- [18] Y.-J. CHIANG AND R. TAMASSIA, *Dynamic Algorithms in Computational Geometry*, Tech. report CS-91-24, Department of Computer Science, Brown University, Providence, RI, 1991.
- [19] P. F. DIETZ, *Optimal algorithms for list indexing and subset rank*, in Proceedings of the Workshop on Algorithms and Data Structures, Springer, London, 1989, pp. 39–46.
- [20] P. F. DIETZ AND R. RAMAN, *Persistence, amortization, and randomization*, in Proceedings of the Second Annual ACM-SIAM Symposium on Discrete Algorithms, 1991, pp. 78–88.
- [21] P. F. DIETZ AND D. D. SLEATOR, *Two algorithms for maintaining order in a list*, in Proceedings of the 19th Annual ACM Symposium on Theory of Computing, 1987, pp. 365–372.
- [22] M. DIETZFELBINGER AND M. AUF DER HEIDE, *A new universal class of hash functions and dynamic hashing in real time*, in Automata, Languages, and Programming: 17th International Colloquium, M. S. Paterson, ed., Springer-Verlag, New York, 1990, pp. 6–19.
- [23] H. IMAI AND T. ASANO, *Dynamic orthogonal segment intersection search*, J. Algorithms, 8

- (1987), pp. 1–18.
- [24] K. V. R. KANTH AND A. K. SINGH, *Efficient dynamic range searching using data replication*, Inform. Process. Lett., 68 (1998), pp. 97–105.
 - [25] D. T. LEE AND C. K. WONG, *Quintary trees: A file structure for multidimensional database systems*, ACM Trans. Database Syst., 5 (1980), pp. 339–353.
 - [26] G. S. LUEKER, *A data structure for orthogonal range queries*, in Proceedings of the 19th Annual Symposium on Foundations of Computer Science, IEEE Computer Society Press, Los Alamitos, CA, 1978, pp. 28–34.
 - [27] E. M. MCCREIGHT, *Priority search trees*, SIAM J. Comput., 14 (1985), pp. 257–276.
 - [28] K. MEHLHORN, *Data Structures and Algorithms: 3. Multidimensional Searching and Computational Geometry*, Springer, Berlin, 1984.
 - [29] K. MEHLHORN AND S. NÄHER, *Dynamic fractional cascading*, Algorithmica, 5 (1990), pp. 215–241.
 - [30] C. W. MORTENSEN, *Fully-Dynamic Orthogonal Range Reporting on RAM*, Tech. report TR-22, IT University of Copenhagen, Copenhagen S, Denmark, 2003.
 - [31] C. W. MORTENSEN, *Fully-dynamic two dimensional orthogonal range and line segment intersection reporting in logarithmic time*, in Proceedings of the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms, ACM, New York, 2003, pp. 618–627.
 - [32] C. W. MORTENSEN, *Generalized Static Orthogonal Range Searching in Less Space*, Tech. report TR-33, IT University of Copenhagen, Copenhagen S, Denmark, 2003.
 - [33] Y. NEKRICH, *Space efficient dynamic orthogonal range reporting*, in Proceedings of the 21st Symposium on Computational Geometry, ACM, New York, 2005, pp. 306–313.
 - [34] M. H. OVERMARS, *The Design of Dynamic Data Structures*, Springer, Berlin, 1983.
 - [35] M. H. OVERMARS, *Efficient data structures for range searching on a grid*, J. Algorithms, 9 (1988), pp. 254–275.
 - [36] M. H. OVERMARS AND J. VAN LEEUWEN, *Dynamic multi-dimensional data structures based on quad- and K -D trees*, Acta Inform., 17 (1982), pp. 267–285.
 - [37] M. SMID, *Range trees with slack parameter*, Algorithms Review, 2 (1991), pp. 77–87.
 - [38] P. VAN EMDE BOAS, *Preserving order in a forest in less than logarithmic time and linear space*, Inform. Process. Lett., 6 (1977), pp. 80–82.
 - [39] P. VAN EMDE BOAS, R. KAAS, AND E. ZIJLSTRA, *Design and implementation of an efficient priority queue*, Math. Systems Theory, 10 (1977), pp. 99–127.
 - [40] D. E. WILLARD, *Predicate-Oriented Database Search Algorithms*, Outstanding Dissertations in the Computer Sciences, Garland Publishing, New York, 1978.
 - [41] D. E. WILLARD, *Log-logarithmic worst-case range queries are possible in space $\Theta(N)$* , Inform. Process. Lett., 17 (1983), pp. 81–84.
 - [42] D. E. WILLARD, *New data structures for orthogonal range queries*, SIAM J. Comput., 14 (1985), pp. 232–253.
 - [43] D. E. WILLARD, *Reduced memory space for multi-dimensional search trees*, in Proceedings of the 2nd Symposium on Theoretical Aspects of Computer Science, Springer, Berlin, 1985, pp. 363–374.
 - [44] D. E. WILLARD, *A density control algorithm for doing insertions and deletions in a sequentially ordered file in good worst-case time*, Inform. and Comput., 97 (1992), pp. 150–204.
 - [45] D. E. WILLARD, *Examining computational geometry, Van Emde Boas trees, and hashing from the perspective of the fusion tree*, SIAM J. Comput., 29 (2000), pp. 1030–1049.
 - [46] D. E. WILLARD AND G. S. LUEKER, *Adding range restriction capability to dynamic data structures*, J. ACM, 32 (1985), pp. 597–617.
 - [47] J. ZHANG, *Density Control and On-Line Labeling Problems*, Tech. report TR481, University of Rochester, Computer Science Department, Rochester, NY, 1993. Ph.D. thesis.